Chris Kluge
KT07w1
Matrikel-Nr.: 20311

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Kfz-Universalgateway

Diplomarbeit

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fachbereich
Informationstechnik und Elektrotechnik

Mittweida, 2009

Betreuer: Prof. Dr.-Ing. Thomas Beierlein
Dipl.-Ing. Stefan Michael Schober
Dipl.-Ing. Marco Sorich

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Kluge, Chris:
Kfz-Universalgateway – Spezifizierung und Implementierung, Mittweida, Hochschule Mittweida (FH), Fachbereich Informationstechnik & Elektrotechnik, Diplomarbeit, 2009

Refarat

Ziel der Diplomarbeit ist es, ist es frei zu programmierendes Universalgateway sowie Diagnoseinstrument für den Webasto-Bus, CAN sowie LIN-Daten zu spezifizieren und auf Basis eines Evaluation Board der Firma NEC zu implementieren. Mit Hilfe dieses Universal-Gateways sollen innerhalb der Prototypenentwicklung (für frühe Funktionsmuster) das Erarbeiten und Testen neuer Algorithmen erleichtert und die Anbindung neuer Heizsysteme an Konzeptfahrzeuge der Fahrzeughersteller realisiert werden können.
Diese Diplomarbeit bildet die Basis für die spätere Hardwareauslegung sowie für die Erstellung der Serien-Software.
Im ersten Teil der Arbeit werden dazu die Grundlagen für die Entwicklung eingebetter System in Kraftfahrzeugen kurz erläutert. Im Anschluss folgt die Anforderungsanalyse, Spezifikation, Implementierung der Softwaremodule sowie deren Qualifizierung.

Chris Kluge
KT07w1
Matrikel-Nr.: 20311

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Mittweida, 30.09.2009 _____

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Inhaltsverzeichnis

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Abkürzungsverzeichnis

ABS	Antblockiersystem
ADC	Analog to Digital Converter
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
DIO	Digital Input/Output
DMA	Direct Memory Access
ECU	electronic control unit (Steuergerät)
EEPROM	electrically erasable programmable read-only memory (elektrisch löschbarer programmierbarer Nur-Lese-Speicher)
ESD	Electrostatic Discharge (Elektrostatische Entladung)
ESP	Elektronisches Stabilitätsprogramm
GPT	General Purpose Timer
HW	Hardware
ICU	Input Capture Unit
ID	Identifikation
ISO	Internationale Organisation für Normung
ISR	Interrupt Service Routine
Kfz	Kraftfahrzeug
KWP2000	Key-Word-Protokoll 2000
LCD	Liquid Crystal Display (Flüssigkristallbildschirm)
LIN	Local Interconnect Network
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
MISRA	Motor Industry Software Reliability Association
MOST	Media Oriented Systems Transport
NM	Network Management
NV-Memory	Non Volatile Memory (nicht flüchtiger Speicher)
OS	Operating System
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
PLL	Phase-locked loop (Phasenregelschleife)
PWM	Pulse Width Modulated (pulsweiten moduliert)
RAM	Random-access memory
RTE	Runtime Enviroment (Laufzeitumgebung)
SPI	Serial Peripheral Interface
SW	Software
UART	Universal Asynchronous Receiver Transmitter
UDS	Unified Diagnostic Services
VDX	Vehicle Distributed Executive
Wbus	Webasto-eigener Datenbus
WDT	Watchdog
XCP	Universal Measurement and Calibration Protocol

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Abbildungsverzeichnis

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Tabellenverzeichnis

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Vorwort

Die vorliegende Diplomarbeit wurde in der Zeit von Mai 2009 bis Oktober 2009 in der Zusammenarbeit zwischen dem Fachbereich Informationstechnik & Elektrotechnik der Hochschule Mittweida (FH) und der Webasto AG in Stockdorf angefertigt.

Ich möchte mich bei all denen bedanken, die an der Entstehung dieser Arbeit mitgewirkt haben, insbesondere bei Herrn Prof. Dr.-Ing. Thomas Beierlein, Herrn Dipl.-Ing. Stefan Michael Schober und Herrn Dipl.-Ing. Marco Sorich für die Betreuung, Durchsicht des Manuskriptes und für die Beantwortung aller Fragen zur Architektur und Implementierung der Software.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Kapitelübersicht

Im **Kapitel 1** erfolgt die Einführung in das Thema dieser Diplomarbeit. Es wird kurz auf die aktuelle Situation innerhalb der der Elektronik-Entwicklung in der Automobilindustrie eingegangen und das Ziel der Diplomarbeit dargestellt.

Kapitel 2 gibt einen Überblick über die wichtigsten Grundlagen und den aktuellen Stand der Technik. Es wird auf den allgemeinen Entwicklungsprozess, Bussysteme, Software-Architekturen und Betriebssysteme von Elektronikkomponenten in der Automobilindustrie eingegangen.

Das **Kapitel 3** befasst sich mit notwendigen Vorabbetrachtungen und der Ermittlung der Anforderungen an das Kfz-Universalgateway.

Im **Kapitel 4** erfolgt der Systementwurf, es wird die Software-Architektur definiert und die Anforderungen an die Hardware werden beschrieben. Ferner werden die Systemfunktionen des Universalgateways spezifiziert.

Kapitel 5 stellt die Implementierung des Betriebssystems, der AUTOSAR-Komponenten, der Kommunikationstreiber und der spezifizierten Anforderungen dar.

Im **Kapitel 6** wird die Qualifizierung der Softwaremodule und des Gesamtsystems beschrieben.

Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und bewertet das Entwicklungsergebnis.

Das **Kapitel 8** gibt mögliche Ausblicke zur Weiterführung des Themas.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

1 Einleitung

Die nachfolgenden Kapitel geben in Bezug auf diese Diplomarbeit einen kurzen Überblick über die Entwicklung der Elektronik im Kraftfahrzeug und den daraus resultierenden Bedarf, ein Universalgateway zu entwickeln. Es werden die Ziele und eine Übersicht über die Inhalte dieser Arbeit dargestellt.

1.1 Situation der Elektronik-Entwicklung in der Automobilindustrie

Am Anfang der Automobilgeschichte bis zum Anfang der 80er Jahre spielten die elektrischen und elektronischen Komponenten im Vergleich zu den mechanischen Komponenten eines Automobils eine untergeordnete Rolle. Seit dem Aufkommen der ersten Steuergeräte, bei denen der mechanische Unterbrecher durch einen kontaktlosen Sensor ersetzt wurde, haben Umfang und Komplexität der elektronischen Komponenten stetig zugenommen. Wurden in der ersten Generation der elektronischen Steuergeräte vornehmlich mechanische Funktionen durch diskrete Elektronik ersetzt (z.B. der Ersatz des mechanischen Unterbrechers durch kontaktlose Sensoren), wurden mit der zweiten Generation der Steuergeräte bereits Mikrokontroller und Software eingesetzt (z.B. der Einsatz elektronisch gesteuerte Einspritzanlagen). Die erste und zweite Generation der Steuergeräte waren noch voneinander unabhängige Systeme mit eigener Sensorik und Aktorik. Mit dem Fortschreiten der Komplexität der Steuergeräte der zweiten Generation begann die Einführung der ersten Datenbusse für die Diagnose (z.B. K-Line).

Bei der dritten Generation der Steuergeräte steht der Einsatz eines Netzwerkes im Mittelpunkt. Durch den Einsatz von verschiedenen Bussystemen wie CAN, LIN, MOST, FLEXRAY im Kraftfahrzeug können alle Steuergeräte auf dieselben Informationen zugreifen. So können redundante Funktionen minimiert und Informationen von Sensoren gemeinsam genutzt werden. Zunehmend werden Funktionen steuergeräteübergreifend realisiert und es entstehen neuartige Komfort- und Sicherheitsfunktionen. So ist zum Beispiel die Realisierung des Elektronischen Stabilitätsprogramms (ESP) erst durch die Kommunikation des Antiblockiersystems (ABS), der Beschleunigungssensoren des Airbags und der Motorelektronik möglich.

Waren anfänglich nur wenige Steuergeräte im Fahrzeug verbaut, so stieg die Anzahl an vernetzten elektronischen Komponenten im Kraftfahrzeug seit der ersten Generation der Steuergeräte stark an.

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

In heutigen Mittelklassefahrzeugen ist die Zahl der Steuergeräte auf etwa 40 untereinander vernetzte Steuergeräte gestiegen. Bei einigen aktuellen Premiumfahrzeugen beläuft sich die Anzahl auf bis zu 70 Steuergeräte mit über 400 Einzelfunktionen. [1]
In Abbildung 1 ist die Entwicklung der Anzahl der Steuergeräte von 1996 bis 2005 bei Fahrzeugen der Firma Volkswagen dargestellt.

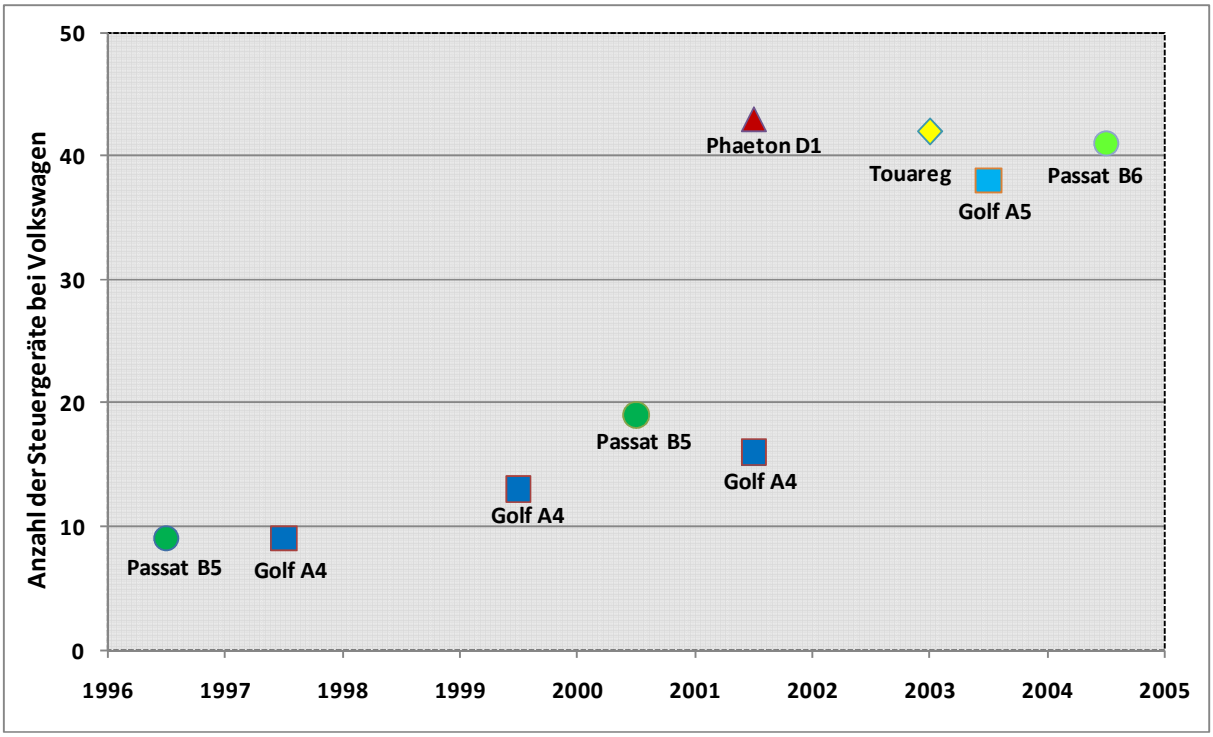


Abbildung 1: Anstieg der Fahrzeugsteuergeräte bei der Volkswagen AG [1]

Bei der Entwicklung der Steuergeräte nimmt die Komplexität der Funktionen bei steigendem Kostendruck und sich verkürzenden Entwicklungszyklen stetig zu.
Die ersten Steuergeräte besaßen noch kein eigenes Betriebssystem. Jeder Software-Entwickler hatte seine eigene Software-Architektur und alle Aufgaben wurden einfach sequentiell abgearbeitet. Mit zunehmender Komplexität wurde der Einsatz eines Multitasking-Betriebssystems unumgänglich. Ferner besteht das Bestreben, eine Architektur zu realisieren, welche die Software von der Hardware des Gerätes entkoppelt, so dass Funktionen steuergeräteübergreifend realisiert werden und durch verschiedene Hersteller entwickelt werden können. Eine solche Entwicklung ist durch eine enge Zusammenarbeit zwischen den Zulieferern und den Automobilherstellern zwingend erforderlich. Nur so können Fehler in der Spezifikation der Funktionen und der Softwareentwicklung frühzeitig erkannt werden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

1.2 Ziel der Diplomarbeit

Die Standheizung in Kraftfahrzeugen wird zunehmend in die Vernetzung moderner Fahrzeuge eingebunden. Informationen, welche bisher über eine direkte Verkabelung erfolgten, werden teilweise nur noch als Signale über den CAN versandt. Für ein korrektes Zusammenspiel der Komponenten ist die Anpassung der Steuerelektronik der Standheizung an das jeweilige Serienfahrzeug in enger Zusammenarbeit mit den Automobilherstellern erforderlich. Eine Möglichkeit, um Fehler bei der Entwicklung von neuen Funktionen und Algorithmen für diese Adaption frühzeitig zu erkennen, ist die Simulation der Funktionen am Computer oder direkt im Kraftfahrzeug. In der Phase der Abstimmung von Pflichten- und Lastenheft bis hin zum ersten Muster stehen jedoch oft noch keine geeigneten Systeme mit passender Software zur Verfügung. Oft werden Fehler der Spezifikation erst beim Systemtest im Fahrzeug aufgedeckt.

Durch den Einsatz von Rapid-Prototyping-Systemen kann diese Lücke geschlossen werden. Algorithmen und Funktionen können mit Hilfe dieser Systeme auf einfache Weise realisiert und unter realen Bedingungen im Fahrzeug erprobt werden. Zukünftig soll innerhalb der frühen Entwicklungsphasen der Standheizung ein solches Rapid-Prototyping-System im Form eines Universalgateways genutzt werden, welches inkompatible Systemkomponenten miteinander verbindet oder Signale generieren bzw. manipulieren kann. Auf diese Weise können Standheizsysteme sehr früh innerhalb der Vernetzung zukünftiger Fahrzeuge erprobt und die erforderlichen Systemparameter ermittelt werden. Es sollen innerhalb des Universalgateways die CAN-Signale des Fahrzeuges ausgewertet und so aufbereitet werden, dass ein über den Webasto-eigenen Datenbus (WBus) angeschlossenes Standheizsystem gesteuert werden kann. Ferner sollen Funktionen zur direkten Ansteuerung von Aktoren, wie z.B. ein Bypassventil im Kühlmittelkreislauf, innerhalb des Universalgateways realisiert werden.

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

In Abbildung 2 ist schematisch die Anbindung des Prototyps einer neuen Standheizung an ein bestehendes Serienfahrzeug über das zu entwickelnde Universalgateway dargestellt.

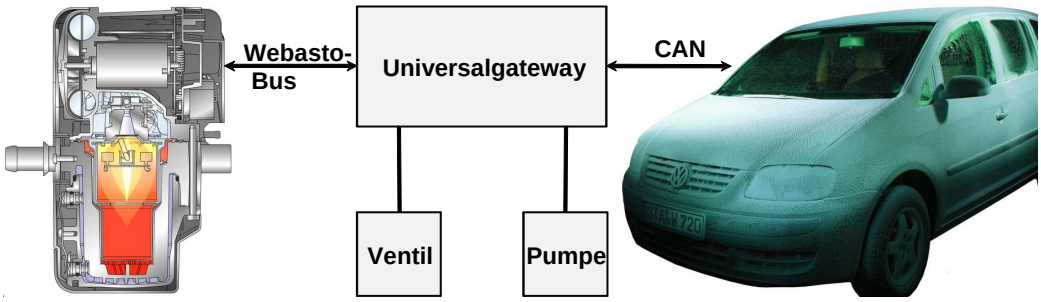


Abbildung 2: Prinzip des Universalgateways

Ziel dieser Diplomarbeit ist es, ein solches Universalgateway als Rapid-Prototyping-Systemen und Diagnoseinstrument für den Webasto-eigenen Bus (WBus), CAN sowie LIN-Daten zu spezifizieren und auf Basis eines Evaluation Board der Firma NEC zu implementieren. Diese Diplomarbeit bildet dabei die Basis für die spätere Hardwareauslegung sowie für die Erstellung der Serien-Software des geplanten Kfz-Universalgateways.

2 Grundlagen und Stand der Technik

In diesem Kapitel wird kurz die gegenwärtige Situation und Vorgehensweise innerhalb der Prototypenphase neuer Standheizsysteme oder Fahrzeugentwicklungen sowie die wichtigsten Grundlagen in Bezug auf diese Diplomarbeit dargestellt.

2.1 Stand der Technik

Die heutige Architektur einer Standheizungs-Software teilt sich in die Bereiche Grundgerät und Applikation auf. Beide Teile der Software sind über eine Abstraktionsschicht verbunden. Das Grundgerät realisiert alle Funktionen, welche zum Betrieb der Standheizung notwendig sind und in jedem Projekt unabhängig von den Kundenanforderungen benötigt werden (z.B. die Brennersteuerung oder das Fehlermanagement). Die Applikation hingegen realisiert die Adaption der Kundenanforderungen an das Grundgerät. Dabei hat jeder Automobilhersteller und teilweise auch jede Fahrzeugplattform eigene Algorithmen und Anforderungen an die Steuerung der Standheizung.

Auf Grund der hohen Anforderungsvielfalt der Automobilhersteller fällt es zunehmend schwerer, neue Entwicklungen in frühen Entwicklungsphasen unter seriennahen Bedingungen zu erproben. Auch stehen in diesen frühen Entwicklungsphasen nur selten bestehende Steuergeräte und eine geeignete Software zur Verfügung, welche mit geringem Aufwand so an die Fahrzeuge angepasst werden können, dass Funktionen und Algorithmen geprüft und Systemparameter ermittelt werden können. Innerhalb der letzten Projekte wurden neben dem Bestreben, aus den verschiedenen Anforderungen herstellerübergreifende Funktionsmodule zu bündeln, zunehmend die Simulationen in Form von Rapid-Prototyping-Systemen im Fahrzeug eingesetzt. Diese Systeme wurden meist direkt in die CAN-Verbindung zwischen dem Fahrzeug und der Standheizung geschaltet, so dass die notwendigen Signale manipuliert werden konnten, um die gewünschten Funktionen zu simulieren. Allen Systemen gemeinsam ist, dass über diese Simulation nicht zu 100% die notwendigen Funktionalitäten dargestellt werden können. Ferner besitzen diese Systeme eine zur Standheizung unterschiedlichen Hardwareplattform und die Softwarearchitektur entspricht nicht der Architektur der Standheizung. So mussten Funktionen, welche dargestellt werden sollen aufwendig portiert werden.

Aus dieser Situation heraus ist die Aufgabenstellung zur Entwicklung eines systemkompatiblen Universalgateways für die Standheizung entstanden, für welches mit dieser Diplomarbeit die Grundlagen und ein erstes Funktionsmuster erarbeitet werden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

2.2 Phasen des Entwicklungsprozesses

Der Entwicklungsprozess eines vernetzten Steuergerätes stellt hohe Anforderungen an die Spezifikation, die Implementierung und die Freigabeprüfungen.
Dieser Entwicklungsprozess ist im Allgemeinen ein iterativer Vorgang, welcher in einem Spiralmodell durch verschiedene Zyklen dargestellt werden kann. [5]

P = Prototyp
A, B, C = Musterstände
S = Serienteil

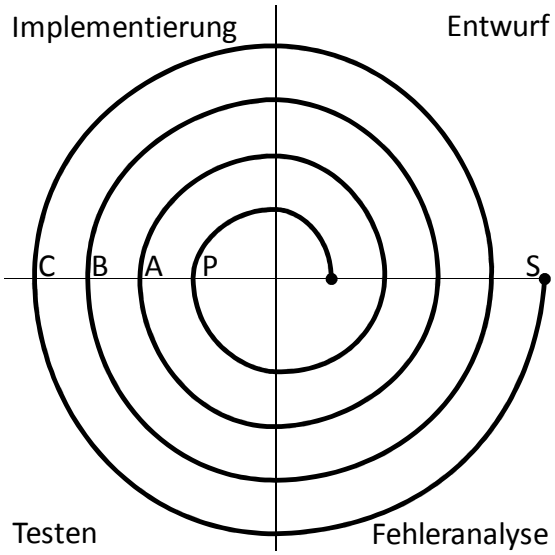


Abbildung 3: Spiralmodell [5]

Die Phasen „Entwurf“, „Implementierung“, „Testen“ und „Fehleranalyse“ werden innerhalb des Entwicklungsprozesses mehrfach durchlaufen (siehe Abbildung 3). Auf diese Weise entstehen Prototyp (P), A-, B- und C-Muster und das endgültige Serienteil (S). Wird innerhalb der Test-Phase eine Abweichung zur Spezifikation festgestellt oder ändert sich die Spezifikation, dann muss ein weiterer Entwicklungszyklus durchlaufen werden. Erst wenn beim Testen keine Abweichungen zwischen der Spezifikation und der Implementierung mehr festgestellt werden kann, erfolgt die Freigabe und das Produkt oder die Baugruppe kann in Serie gehen.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Jeder Entwicklungszyklus folgt dabei der Vorgehensweise des V-Modells. Das V-Model ist eine abstrakte Darstellung der Projektelemente in V-Form (siehe Abbildung 4). Das V-Modell unterteilt die Entwicklungsphasen in zwei Bereiche – dem Entwurfszweig (linke Seite) und dem Testzweig (rechte Seite). Dabei beinhaltet der Entwurfszweig die Phasen Analyse-Spezifikation und Implementierung. Der Testzweig realisiert die Test-Phasen.

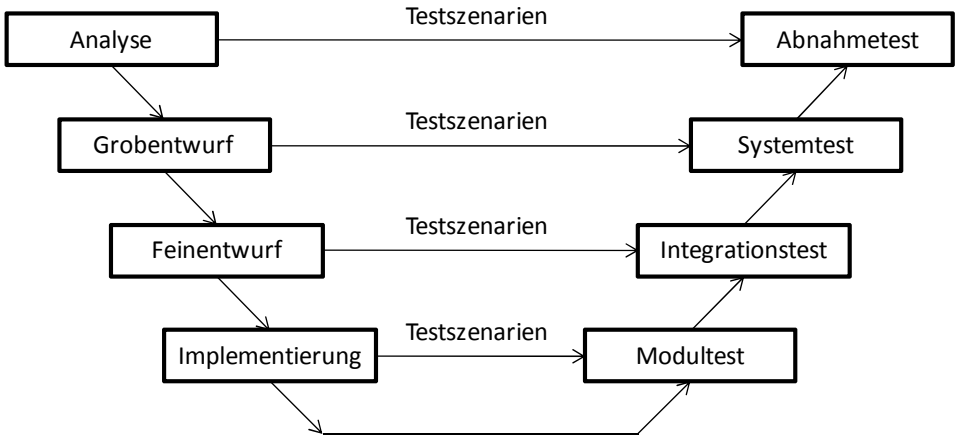


Abbildung 4: V-Modell [5]

Aus jeder Phase des Entwurfszweiges ergeben sich dabei die Anforderungen der Testfälle für die jeweiligen Phasen des Testzweiges. Ferner findet innerhalb jeder Phase des Entwurfszweiges ein Selbsttest statt, in welchem geprüft wird, ob die Anforderungen der übergeordneten Schicht erfüllt werden. Dieser Selbsttest entscheidet über das weitere Vorgehen innerhalb des Entwicklungsprozesses. Das Ende des Entwicklungsprozesses ist erreicht, wenn der Abnahmetest (letzte Phase des Testzweiges) das Ergebnis „in Ordnung“ ergibt. Gibt es noch Abweichungen, müssen die Phasen des V-Modells ein weiteres Mal durchlaufen werden (siehe Spiralmodell).
Zur Reduzierung des Entwicklungsrisikos werden innerhalb der Spezifikationsphase komplexe Funktionen mit speziellen Programmen simuliert und zum Teil mittels Rapid-Prototyping-Systemen, wie dieses Universalsteuergerät, direkt im Fahrzeug validiert.

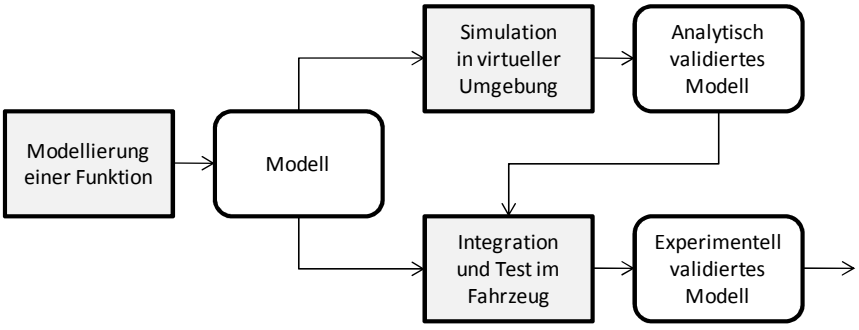


Abbildung 5: Simulations- und Rapid-Prototyping-Schritte [7]

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

2.3 Bussysteme

Für die Vernetzung von Informationen zwischen den Steuergeräten eines Fahrzeugs werden Bussysteme eingesetzt. In der Kraftfahrzeugtechnik werden hohe Anforderungen an die Ausfallsicherheit der verwendeten Busse gestellt. Es muss sichergestellt werden, dass die Empfänger einer Nachricht alle Informationselemente sicher identifizieren können. Dabei müssen fehlerhafte Übertragungen erkannt und korrigiert werden können.
Die Anwendungsgebiete für Bussysteme innerhalb eines Kraftfahrzeugs lassen sich dabei grob in drei Bereiche unterteilen. [3]

1. High Speed Systeme für Echtzeit-Steuerungsaufgaben
   Dazu gehören z.B. die Motorsteuerung, die Regelung von Bremse, Getriebe und Fahrwerk.
2. Low Speed Systeme zur Kabelbaumvereinfachung
   Über einen solchen Bus werden einfache Aktoren wie Lampen, Fensterheber oder Spiegelmotoren gesteuert.
3. Infotainment-Bussysteme
   Diese dienen zur Kommunikation zwischen Infotainment-Systemen wie Navigation, Video, Audio und Telefon. In dieses Bussystem müssen zum Teil hohe Datenmengen z.B. zwischen dem CD-Wechsler und dem verteilten Audio-System transportiert werden.

Entsprechend der genannten Anforderungen kommen im Fahrzeug verschiedene Bussysteme zum Einsatz.

	Datenmenge	**Anforderungen an Fehlersicherheit**	**geeignetes Bussystem**
High Speed Systeme	hoch	hoch bis sehr hoch	High Speed CAN / Flexray
Low Speed Systeme	gering	mäßig	Low Speed CAN / LIN, Webasto-Bus
Infotainment-Systeme	sehr hoch	gering	MOST

Tabelle 1: Anforderungen an Netzwerke im Fahrzeug

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Allen im Fahrzeug verwendeten Bussen gemein ist, dass diese die Daten bitweise seriell übertragen. Nachfolgend werden die Bussysteme CAN, LIN und der Webasto-Bus kurz erläutert.

2.3.1 Der CAN-Bus

Der CAN (Controller Area Network) ist derzeit das im Fahrzeug am häufigsten eingesetzte Bussystem für Low-Speed- und High-Speed-Anwendungen. Er wurde in der zweiten Hälfte der 80er Jahre von der Firma Bosch entwickelt, um die wachsenden Anforderungen der Fahrzeugtechnik und die zunehmende Menge an Informationen bewältigen zu können [3].
Der CAN-Bus ist ein Linien-Bus, an welchen die einzelnen Steuergeräte mit kurzen Stichleitungen angeschlossen sind. Das Übertragungsmedium ist eine verdrillte Zwei-Draht-Leitung mit einem Wellenwiederstand von 120 Ohm (Abbildung 6). Zur Vermeidung von Reflexionen wird der CAN-Bus an den beiden Enden mit einem Widerstand abgeschlossen.

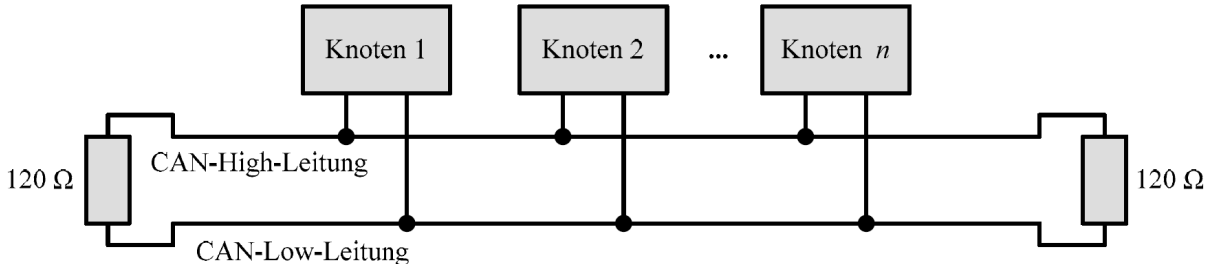


Abbildung 6: CAN-Bus mit Abschlusswiderständen [8]

Von der physikalischen Ebene des CAN existieren derzeit mehrere Varianten, welche sich hauptsächlich in den Signalpegeln und der maximalen Übertragungsgeschwindigkeit unterscheiden (Abbildung 7). In der Fahrzeugtechnik kommen hauptsächlich die Varianten nach ISO 11848-2 (High-Speed-CAN bis 1Mbit/s) und nach ISO 11848-3 bzw. ISO 11519 (Low-Speed-CAN bis 125kbit/s) zum Einsatz.

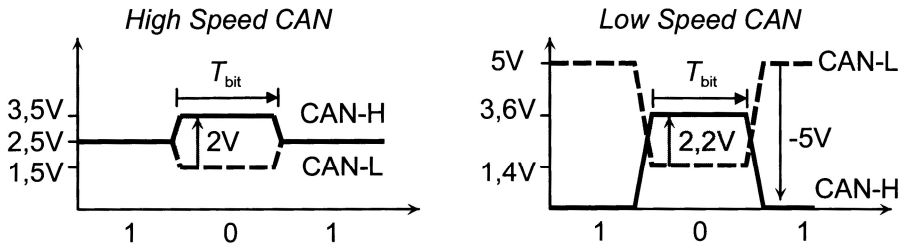


Abbildung 7: Signalpegel bei High- und Low-Speed-CAN [3]

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Komfort-Systeme wie z.B. die Klimaanlage sind nicht sicherheitsrelevant und übertragen meist nur geringe Datenmengen. Für diese Gruppe von Steuergeräten wird daher meist der Low-Speed-CAN eingesetzt. Motor- und Fahrwerks-Komponenten sind sicherheitsrelevant und übertragen in kurzer Folge hohe Datenmengen. Aus diesem Grund kommt hier zur Vernetzung der High-Speed-CAN zum Einsatz.
Derzeit ist ein Bestreben in der Automobilindustrie zu erkennen, nur noch ein einheitliches CAN-Bus-System einzusetzen. Der Low-Speed-CAN wird demnach in zukünftigen Fahrzeugen nicht mehr zum Einsatz kommen.
Der CAN ist ein Multimaster-Bussystem. Alle Busteilnehmer sind gleichberechtigt. Jedes Steuergerät darf senden, wenn der Bus für min. 3 Bitzeiten frei ist. Beim Buszugriff wird das CSMA/CA-Verfahren (Carrier Sense Multiple Access with Collision Avoidance) zur Auflösung von Kollisionen verwendet. Jedem Informationspaket (Botschaft) ist eine eindeutige 11 Bit (Standard) oder 29 Bit (Extended) lange Identifikation (ID) zugeordnet. Die Priorität der gesendeten Botschaften wird dabei durch deren Message-Identifier gekennzeichnet (niedrige Zahl = hohe Priorität). Kommt es beim Zugriff auf den Bus zu einer Datenkollision, so wird die Botschaft mit geringerer Priorität zerstört. Der Sender dieser Botschaft muss sofort das Senden einstellen und warten, bis der Bus wieder frei ist. Die höher priorisierte Botschaft bleibt erhalten und der Sender dieser Botschaft darf weitersenden. In Abbildung 8 ist die Auflösung einer Kollision auf dem CAN dargestellt.

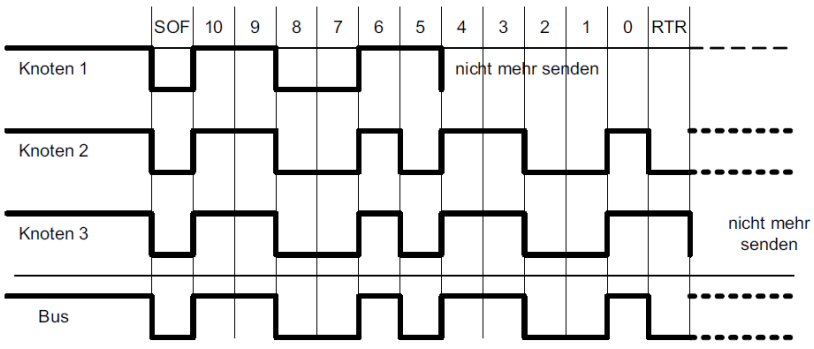


Abbildung 8: Auflösung der Kollision zweier CAN-Botschaften [6]

Auf dem CAN werden Informationen nicht, wie vom Ethernet bekannt, an einen bestimmten Empfänger versandt. Die Informationen auf dem CAN werden im Regelfall als Broadcast-Nachricht allen Teilnehmern zur Verfügung gestellt (Producer-Consumer-Modell). Jeder CAN-Knoten filtert die für ihn benötigten Botschaften aus dem Datenstrom heraus. Alle anderen empfangenen Botschaften werden ignoriert. Die einzelnen CAN-Botschaften werden typischerweise zyklisch versandt. Bei einigen Botschaften wird dieser Zyklus unterbrochen,

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

wenn sich eine besondere Information innerhalb der CAN-Botschaft ändert, auf die andere Teilnehmer sofort reagieren sollen. In diesem Fall wird die Botschaft sofort auf dem Bus aktualisiert. Jede CAN-Botschaft kann maximal 8 Byte Nutzdaten übertragen.
Die Informationen der Nutzdaten sind meist bitweise fest codiert (siehe Tabelle 2). So ist für die Nutzung von CAN-Daten nur der Identifier der Botschaft, das Bit, bei dem die Information beginnt, Signallänge, Offset und Skalierung des Signals notwendig.

Identifier	Botschaftslänge (DLC)	Botschaft	Start-Bit	Länge (Bit)	Signal	Offset	Skalierung	Sender/Empfänger						Bemerkung
								Motor	Kombi	Getriebe	ABS	Airbag	Gateway	
0x100	3	Motor_1	0	14	Motordrehzahl	0	1	S	E	E	E		E	0…16383 U/min 16384 = Fehler
			14	1	Öldruck	0	1	S	E				E	0 = n.i.O. 1 = i.O.
			15	1	Wegfahrsperre	0	1	S	E				E	0 = nicht berechtigt 1 = berechtigt
			16	8	Kühlwassertemperatur	-40	0,75	S	E				E	-40℃ … 150,5℃ 151,25℃ = Fehler

Tabelle 2: Beispiel-Signaldefinition einer CAN-Botschaft

Zur Erkennung gestörter oder verfälschter Botschaften sind auf der physikalischen Ebene des CAN-Protokolls Grundmechanismen wie Bitmonitoring, Überwachung des Telegrammformates, Checksummen, Bitstuffing und Acknowledgement integriert.
Neben diesen Grundmechanismen bietet die Ausführung des Busses als Zweidrahtleitung zusätzlich Redundanzen im Fehlerfall. Wird eine der beiden Leitungen unterbrochen oder kurzgeschlossen, so kann das Signal weiterhin übertragen werden. Im Fehlerfall wird der CAN wie ein Ein-Draht-System genutzt und die Fahrzeugmasse bildet den Bezugspunkt.
Die dargestellten Redundanzen und Maßnahmen zur Fehlersicherung ermöglichen den Einsatz des CAN in sicherheitskritischen Bereichen.

2.3.2 Der LIN-Bus

Ende der 1990er Jahre wurde der LIN-Bus als kostengünstige Alternative zum Low-Speed-CAN für die Realisierung einer einfachen Sensor-Aktor-Kommunikation und als Ersatz für individuelle Sub-Bus-Systeme entwickelt.

Der LIN-Bus ist ein Single-Master-Multi-Slave-System, welcher als Übertragungsmedium mit einer Leitung auskommt. Als Bezugspotential dient die Masseverbindung über die Fahrzeug-

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

karosserie. Maximal können 16 Teilnehmer an den bis zu 40m langen LIN-Bus angeschlossen werden. Die physikalische Ebene des LIN entspricht dem der K-Line, welcher zum Zeitpunkt der Entwicklung des LIN als Diagnose-Schnittstelle weit verbreitet war. Auf diese Weise standen sehr früh geeignete Transceiver für den LIN zur Verfügung. Das asynchrone byteorientierte Protokoll basiert auf dem bekannten UART-Protokoll (8 Daten-Bits mit je einem Start- und Stoppbit ohne Parität), welches von nahezu jedem Mikrocontroller unterstützt wird. Die Übertragungsgeschwindigkeit auf dem LIN liegt im Bereich von 1 bis 20 kbit/s. Im Kraftfahrzeug werden meist Übertragungsgeschwindigkeiten von 9,6 kbit/s und 19,2 kbit/s genutzt. Auf Grund der geringen Übertragungsgeschwindigkeit und des Master-Slave-Verfahrens wird der LIN ausschließlich als Sub-Bus in CAN-Netzwerken eingesetzt. Das CAN-Steuergerät fungiert dabei als Gateway und bildet den Master für die am LIN angeschlossenen Systeme (Abbildung 9).

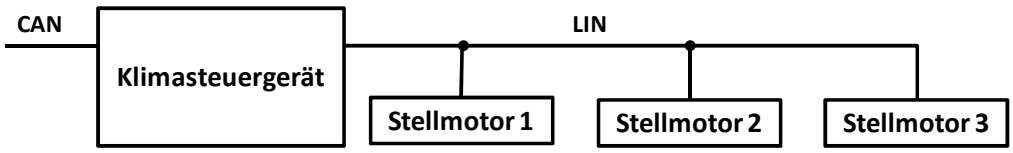


Abbildung 9: Beispiel für einen LIN als Sub-Bus

Anders als beim CAN-Bus sind beim LIN zur Erkennung von Übertragungsfehlern nur wenige Mechanismen integriert (Rücklesen der Bussignale, Prüfsumme, Antwortzeitüberwachung). Ferner gibt es keinerlei Verfahren um Übertragungsfehler selbstständig korrigieren zu können. Aus diesem Grund wird der LIN nur bei Steuergeräten eingesetzt, welche keine sicherheitskritischen Funktionen realisieren. Die Daten werden wie beim CAN inhaltsbezogen adressiert. Jede LIN-Botschaft kann 8 Datenbytes übertragen und besitzt einen eindeutigen Identifier (6 Bit). Somit sind lediglich 64 verschiedene LIN-Botschaften je Sub-Bus möglich. Die Datenübertragung selbst erfolgt kollisionsfrei. Das Master-Steuergerät fragt zyklisch von jedem LIN-Knoten die Daten ab. Die Reihenfolge und die Wiederholperiode jeder LIN-Botschaft sind innerhalb des Master-Steuergerätes in einer Botschaftstabelle (Schedule Table) definiert.

2.3.3 Der Webasto-Bus

Der Webasto-Bus (WBus) wurde zur direkten Kommunikation der Webasto-Standheizung mit den Bedienelementen und zur Diagnosekommunikation entwickelt. Wird eine Standheizung werksseitig bei einem Automobilhersteller verbaut, dann dient das Steuergerät der Standhei-

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

zung als Gateway und der WBus meist als herstellerspezifischer Sub-Bus zur Kommunikation mit der Funkfernbedienung (Abbildung 10).

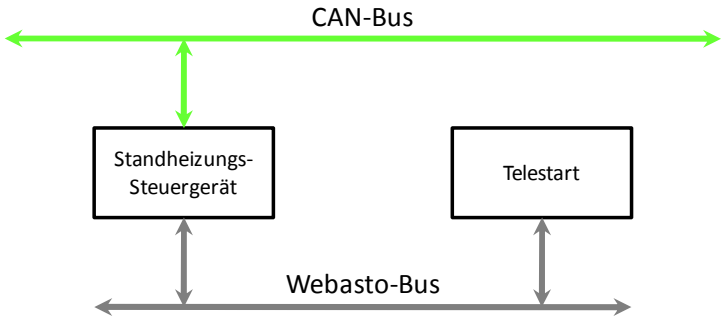


Abbildung 10: Webasto-Bus als Sub-Bussystem [19]

Die physikalische Schicht des Webasto-Busses entspricht dem LIN (Ein-Draht-Bussystem mit Fahrzeugmasse als Bezugspunkt). Der WBus ist jedoch ein Multi-Master-Bus mit maximal 8 Teilnehmern. Jedem Busteilnehmer ist eine feste Adresse zugeordnet, welche gleichzeitig die Priorität des Teilnehmers auf dem Bus definiert. Die Bit-Übertragungs-Ebene ist wie beim LIN byteorientiert und basiert auf dem bekannten UART-Protokoll. Eine Kollision von Telegrammen auf dem Bus wird durch das CSMA/CA-Verfahren (Carrier Sense Multiple Access with Collision Avoidance) aufgelöst. Die Datenübertragung erfolgt adressorientiert nach dem Client-Server-Modell. Jedes Daten-Telegramm enthält in einem Byte die 4-Bit-Adresse für den Empfänger und der Absender des Telegramms. Zur Erkennung einer Kollision auf dem Bus wird die gesendete Information vom Sender zurückgelesen und mit den zu sendenden Daten verglichen. Die Absicherung der übertragenen Daten erfolgt über eine Checksumme am Ende des Telegramms und durch die Überprüfung der Telegrammlänge. Dazu wird in jedem Telegramm die Anzahl der übertragenen Bytes codiert. Anders als beim CAN oder LIN sind die Informationen auf dem WBus nicht fest bitweise codiert. Die Daten werden über ein Kommando funktionsspezifisch übertragen (ähnlich wie beim Diagnoseprotokoll KWP2000 oder UDS). Zur Abfrage von Informationen (z.B. zur Diagnose des Systems) sendet der Client eine Anfrage (Request) an den Server. Dieser antwortet mit den geforderten Daten (Response). Für das Steuern von Funktionen (z.B. Starten der Standheizung) wird ein Steuerkommando vom Client an den Server gesandt. Dieser antwortet mit einer Statusmeldung (z.B. Standheizen gestartet).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

2.3.4 Netztopologie

Die Bustopologie bestimmt wesentlich die Eigenschaften des Netzwerkes. Die wichtigsten Varianten der Netztopologie sollen hier kurz vorgestellt werden.

Die Verbindung aller Teilnehmer über einen gemeinsamen Bus ist das wesentliche Merkmal der **Bus-Topologie**. Die Informationen auf dem Bus stehen allen Netzknoten ohne ein weiteres Routing zur Verfügung. Die Netzknoten teilen sich die verfügbare Bandbreite des Übertragungsmediums. Bei einem Multi-Master-Bus besitzen alle Teilnehmer Verfahren zur Erkennung und Vermeidung von Daten-Kollisionen erforderlich. Ein Ausfall eines Netzknotens hat nur geringe Auswirkungen auf den Bus. Alle weiteren Teilnehmer können weiter miteinander kommunizieren.

Bei der **Stern-Topologie** sind alle Teilnehmer des Busses über eine Punkt-zu-Punkt-Verbindung an einen zentralen Knoten angeschlossen. Informationen, welche andere Teilnehmer benötigen, müssen über den Sternpunkt weitergeleitet werden. Durch die exklusive Anbindung steht jedem Teilnehmer die volle Bandbreite des Übertragungsmediums zur Verfügung. Ein Netzwerk in Stern-Topolgie kommt auch zum Einsatz, wenn verschiedenartige Netze miteinander gekoppelt werden sollen oder wenn Sicherheitsrelevante voneinander zu trennen sind. Ein Ausfall des Sternpunktes führt jedoch zum Totalausfall des Netzwerkes bzw. der Verbindung der Netze untereinander.

Die **Ring-Topologie** ist durch eine geschlossene Kette von Punkt-zu-Punkt-Verbindungen zwischen den Netzkonten gekennzeichnet. Jeder Teilnehmer ist mit genau zwei anderen Teilnehmern verbunden. Die gesendeten Informationen werden dabei von Netzknoten zu Netzknoten weitergereicht, bis diese den Empfänger erreicht haben. Dadurch entstehen recht hohe Durchlaufzeiten für eine gesendete Information.

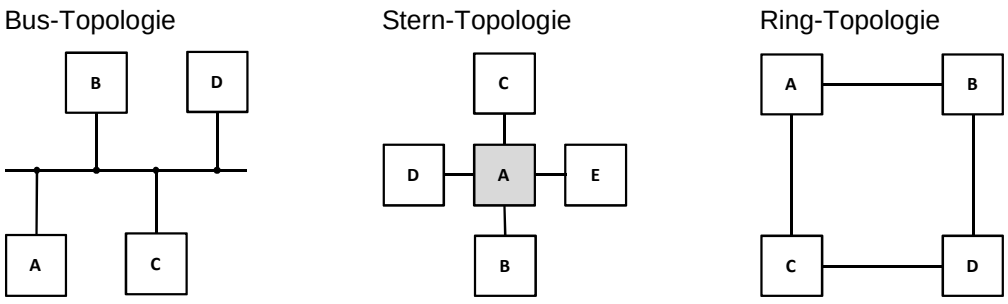


Abbildung 11: Netz-Topologien

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Abbildung 11 zeigt den schematischen Aufbau der erläuterten Netz-Topologien.
In Kraftfahrzeugen werden Stern- und Bus-Topologie miteinander kombiniert. Der CAN-Bus ist als Linienbus ausgeführt (Bus-Topologie), bei dem die Steuergeräte über kurze Stichleitungen angeschlossen sind. Zur Trennung von sicherheitskritischer und -unkritischer Datenkommunikation sowie zur Reduzierung der Buslast werden die CAN-Busse aufgeteilt. Damit die Informationen der einzelnen Steuergeräte auch Teilnehmern an anderen CAN-Bussen zur Verfügung stehen, werden einzelne Signale und Botschaften über ein Gateway verteilt (Stern-Topologie). Durch diese Netztopologie wird ferner sichergestellt, dass bei einer Störung eines Busses (z.B. ein Kurzschluss der Leitungen) der Fehler lokal bleibt. Einzelne Aktoren und Sensoren der CAN-Steuergeräte (z.B. Fensterheber) können an die Steuergeräte über einen LIN als Sub-Bus angeschlossen werden. Das CAN-Steuergerät fungiert dabei als Gateway für die LIN-Daten und Diagnose-Kommunikation. Abbildung 12 zeigt den beispielhaften Aufbau eines Netzwerkes im Kraftfahrzeug.

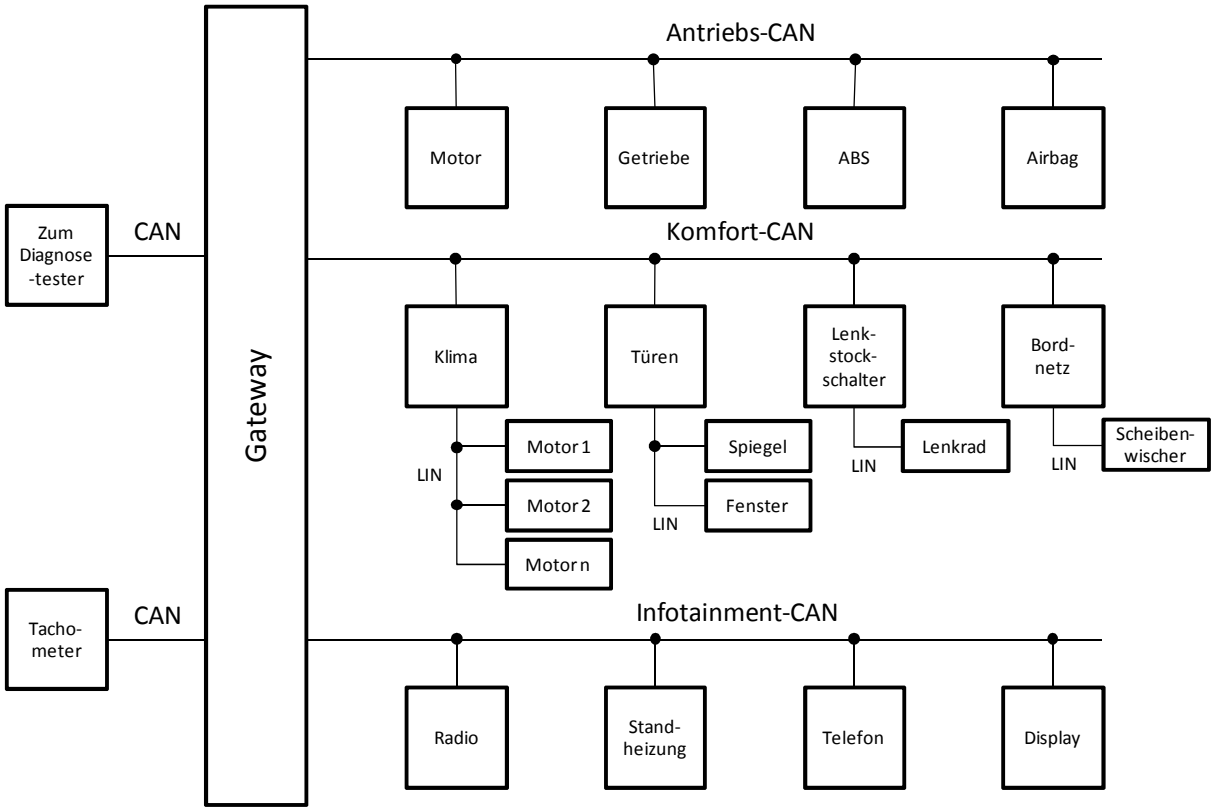


Abbildung 12: Beispiel der Vernetzung von Steuergeräten im Fahrzeug

Innerhalb der Kraftfahrzeugelektronik wird die Ring-Topologie von Netzwerken beim MOST- (Media Oriented System Transport) eingesetzt. Das MOST-System ist für Telematik und

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Multimedia-Anwendungen konzipiert, welche eine hohe Bandbreite erfordern. Als Übertragungsmedium zwischen den Steuergeräten wird ein Kunststoff-Lichtwellenleiter genutzt.
Das MOST-System wird innerhalb des Universal-Steuergerätes nicht verwendet.

2.3.5 Das Gateway

Wie schon im vorangegangenen Kapitel erwähnt, wird als Sternpunkt zwischen den Netzwerken ein zentrales Steuergerät eingesetzt, welches als Gateway die Daten zwischen den verschiedenen Bussen austauscht (Abbildung 13). Gateways arbeiten oberhalb der Schicht 7 des OSI-Referenzmodells. Auf diese Weise können Gateways Informationen völlig verschiedener Bussysteme oder Bussysteme unterschiedlicher Geschwindigkeiten übertragen.

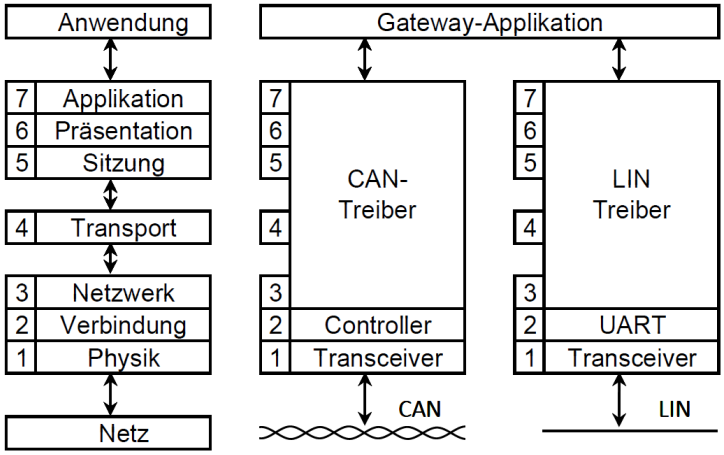


Abbildung 13: OSI-Modell eines Gateways zwischen verschiedenartigen Bussen

Werden über ein Gateway mehrere identische Netzwerke (z.B. 2x High-Speed-CAN mit 500 kbit/s) miteinander verbunden (siehe Abbildung 14), so gibt es die Möglichkeit auf der Schicht 3 des OSI-Referenzmodells, eine Routing-Funktion für Informationen zu realisieren, welche ohne weitere Bearbeitung weitergeleitet werden sollen (Botschafts-Routing). Dies entlastet die eigentliche Gateway-Applikation. Auf der Schicht 7 bzw. innerhalb der Gateway-Applikation werden dann nur noch diejenigen Informationen bearbeitet, welche für das Ziel-Netz angepasst werden müssen.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

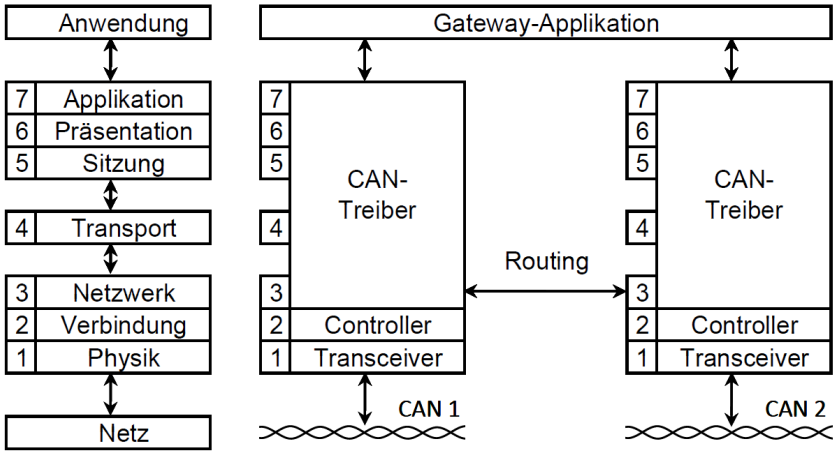


Abbildung 14: OSI-Modell eines Gateways zwischen identischen Bussen

Mit einem Gateway ist es möglich, Informationen neu zu generieren, zusammenzufassen oder wegzulassen, so wie es innerhalb des Ziel-Netzwerkes benötigt wird (Signal-Routing). Das Routing der Informationen (bzw. ganzer Frames auf Schicht 3) erfolgt dabei anhand einer fest konfigurierten Routing-Tabelle. Innerhalb der Entwicklungsphase eines Fahrzeugnetzwerkes wird festgelegt, welches Steuergerät sich in welchem Teilnetzwerk befindet. Ein dynamisches Anpassen des Routings ist daher nicht erforderlich.

Für das Gateway ergeben sich somit folgende Möglichkeiten zum Austausch der Informationen zwischen den Netzwerken:

- Umsetzung der kompletten Botschaft auf ein anderes Netzwerk (Botschafts-Routing)
  - Routing auf Schicht 3 des OSI-Modells als „Schneller Durchreicher“
  - Routing innerhalb der Applikation mit Anpassung der Zykluszeiten
- Zusammenfassen von Informationen verschiedener Botschaften zu einer neuen Botschaft zur Reduzierung der Buslast. Dabei werden nur die im Zielnetz benötigten Informationen weitergeleitet. (Signal-Routing)
- Manipulieren einzelner Informationen einer Botschaft
  z.B. wenn im Zielnetz eine andere Skalierung eines Signals erwartet wird
- Erzeugen neuer Informationen
  z.B. eine Statusinformation über den Zustand des Netzwerkes

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

2.4 Software-Architektur

Die Software der ersten Steuergeräte im Kraftfahrzeug war noch sehr einfach aufgebaut. Ein Betriebssystem war nicht erforderlich. Es wurden in einer Endlosschleife die Sensordaten abgefragt, Ergebnisse berechnet und die Aktoren angesteuert. Ferner gab es keine einheitliche Software-Architektur. So wurde direkt aus dem Programm ohne eine Abstraktionsschicht auf die Hardware zugegriffen. Implementierte Funktionen konnten so nur schwer wiederverwendet oder auf ein anderes System portiert werden.

Grundlage für das Universalgateway ist die Softwarearchitektur nach AUTOSAR mit einem OSEK-Betriebssystem als Basis sein. In den nachfolgenden Kapiteln wird daher kurz auf diese Punkte eingegangen.

2.4.1 Grundlegende Software-Architektur

Mit der zunehmenden Komplexität der Steuergerätefunktionen erwies sich die hardwarenahe Programmierung als nachteilig. Es wurde die Einführung eines recourcenschonenden und leistungsfähigen Echtzeit-Betriebssystems sowie einer einheitlichen Steuergerätearchitektur notwendig. Im Grundsatz werden Basisaufgaben, welche unabhängig von der Art des Steuergerätes sind, auf eine eigene zentrale Ebene, dem Betriebssystem, ausgelagert. Die Hardware wird nicht mehr direkt durch die Applikation angesteuert. Über Funktionsaufrufe werden die zu erledigenden Aufgaben dem Betriebssystem mitgeteilt.

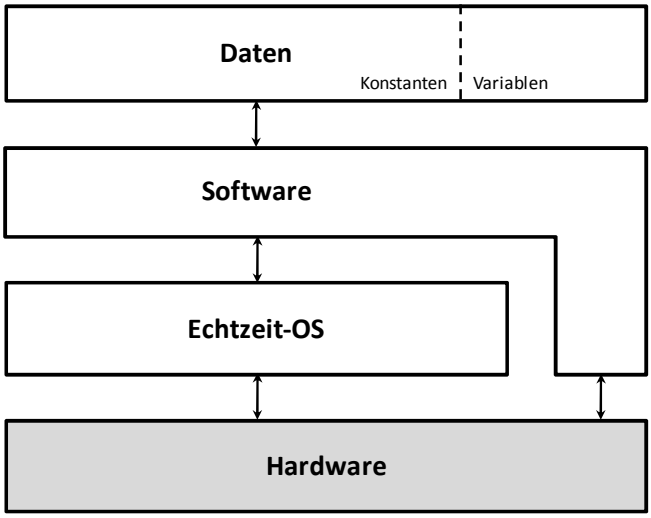


Abbildung 15: Grundlegende Struktur einer Steuergeräte-Software [6]

Heutige Steuergeräte nutzen die in Abbildung 15 dargestellte Grundstruktur mit einem Echtzeit-Betriebssystems als Basis, welches die einzelnen Tasks steuert und überwacht sowie einer separaten Schicht zur Abstraktion der Hardware und zur Bereitstellung von Dienstprogrammen. Zusammen bilden diese die Basis-Software, auf der die individuelle Applikation aufbaut.

2.4.2 OSEK/VDX

Die Aufgabe eines Betriebssystems ist es, eine Ebene zur einfachen Nutzung von Strukturen und Funktionen zwischen der Hardware und der Applikation zur Verfügung zu stellen. [10]

Innerhalb der Automobilelektronik hat sich OSEK/VDX (**O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug) als Betriebssystem durchgesetzt. Dabei stellt OSEK/VDX lediglich die Spezifikation des Betriebssystems dar und ist seit 2006 als Standard in der ISO 17356 genormt. Die auf Basis dieses Standards sind von verschiedenen Softwareherstellern zueinander kompatibler Echtzeitbetriebssysteme entstanden.

OSEK/VDX besteht aus vier unabhängigen Hauptkomponenten:

- OSEK-OS (OS-Kern für ereignisgesteuerte Systeme)
- OSEK-Time (OS-Kern für zeitgesteuerte Systeme)
- OSEK-COM (interne und externe Kommunikation)
- OSEK-NM (Netzwerkmanagement)

Kern des Betriebssystems bildet das Task-Modell. Beim OSEK-OS ist dieses durch Prioritäten gesteuert. Beim OSEK-Time erfolgt die Steuerung zeitgesteuert. Es können OSEK-OS und OSEK-Time parallel in einem System angewendet werden.
Für eine effektive Konfiguration des Betriebssystems ist es erforderlich, die Aufgaben, welche ein System abzuarbeiten hat, in einzelne Tasks zu zerlegen und die Gewichtung, mit der diese Einzelaufgaben zu bearbeiten sind, zu ermitteln. Aufgaben mit gleicher Gewichtung können dabei zu einem Task, welcher dann sequentiell abgearbeitet wird, zusammengefasst werden.

Zur Steuerung der Basic-Tasks kennt OSEK drei Zustände (Suspended, Ready und Running). Wurde ein Task aktiviert, dann wechselt dieser in den Zustand „Running“. Anhand der Prioritäten, welche den Tasks zugewiesen wurden, entscheidet der Scheduler, welcher startbereite Task als nächstes in den Zustand „Running“ versetzt wird.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

OSEK/VDX kennt neben den einfachen Basic-Tasks noch Extended-Tasks, welche zusätzlich über Events auf äußere Ereignisse warten können. Extended-Tasks benötigen daher zusätzlich den Zustand „Waiting“ (Abbildung 16).

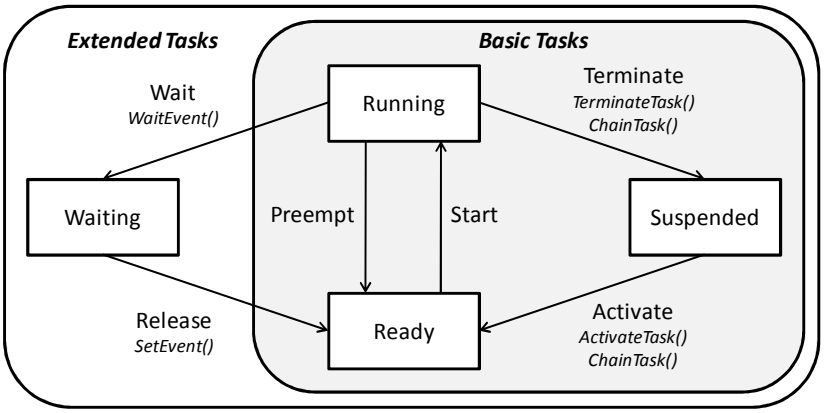


Abbildung 16: Zustandsmodell für OSEK / VDX-Tasks [3]

Jeder Task des OSEK-OS besitzt eine feste Priorität. Befinden sich mehrere Prozesse im Zustand „Ready“, dann wird der Task mit der höchsten Priorität gestartet.
Als Scheduling-Strategie kennt das OSEK-OS drei Strategien.

- nicht präemptiv (ein aktiver Task kann nicht unterbrochen werden)
- präemptiv (ein Task höherer Priorität unterbricht einen aktiven Task)
- gemischt (es existieren präemptive und nicht präemptive Tasks parallel)

Zur Realisierung eines möglichst recourcenschonenden Betriebssystems werden alle Eigenschaften des OSEK/VDX innerhalb der Entwicklungsphase fest konfiguriert. Zur Laufzeit ist es daher nicht möglich, neue Tasks zu erzeugen, deren Priorität zu ändern oder die Scheduling-Strategie zu ändern.

2.4.3 AUTOSAR

Hinter AUTOSAR (Automotive Open System Architecture) steht das Bestreben, die Software von der Hardware weitestgehend zu entkoppeln. Die Software soll dabei aus Funktionsmodulen und Softwarekomponenten bestehen, welche unabhängig voneinander durch verschiedene Hersteller entwickelt werden können. Nach der individuellen Konfiguration der Module erfolgt weitestgehend automatisch das Zusammenfügen der Komponenten zu einem Projekt.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

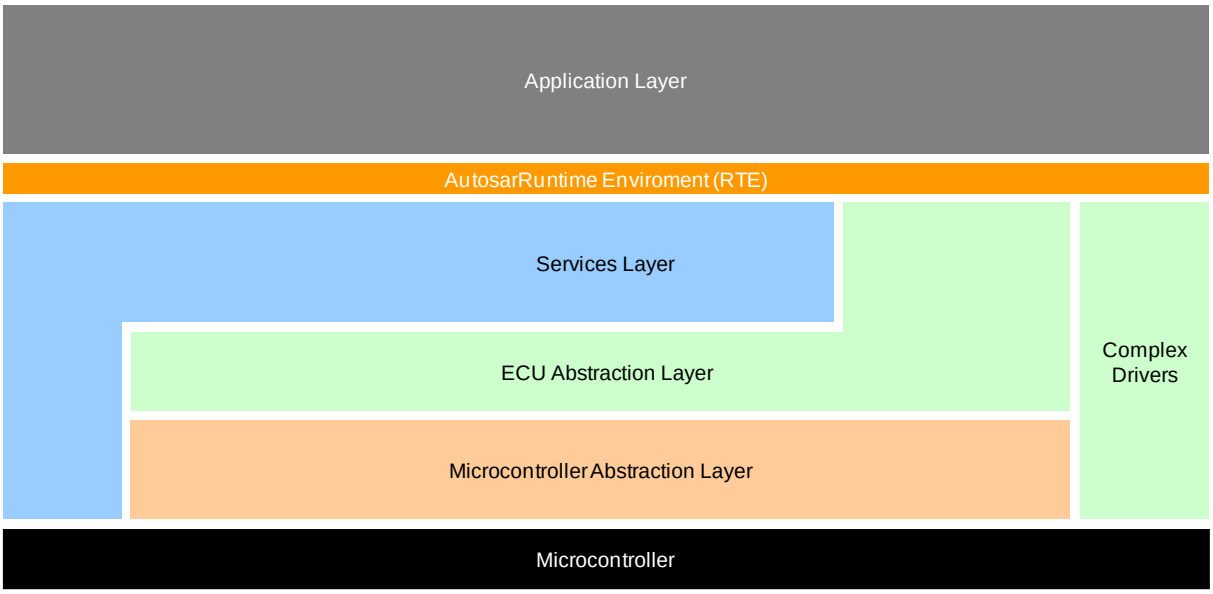


Abbildung 17: Überblick über die AUTOSAR-Software-Layer [13]

Die Abstraktion der Hardware geht im Idealfall so weit, dass es für die Applikation egal ist, ob ein benötigter Service auf diesem oder einem anderen Steuergerät realisiert wurde.
Abbildung 17 zeigt den allgemeinen Aufbau der AUTOSAR-Software-Architektur, welches auch Grundlage für das Universalgateway ist.

Alle Softwaremodule unterhalb der Applikation bilden die **Basis-Software**, welche sich in die folgenden Abstraktions-Schichten aufteilt.

Der **Microcontroller Abstraction Layer** bildet die unterste Schicht der Basis-Software. Dieser Layer enthält Low-Level-Treiber für den Zugriff auf die interne Peripherie des Microcontrollers. Es wird eine Unabhängigkeit vom eingesetzten Microcontroller erreicht.

Innerhalb des **ECU Abstraction Layers** befinden sich die Interfaces für die Module des Microcontroller Abstraction Layers. Ferner befinden sich innerhalb dieser Schicht die Treiber für den Zugriff auf die externe Peripherie. Durch diese Abstraktionsschicht wird eine Unabhängigkeit von der Steuergeräteelektronik erreicht.

Der **Service Layer** bildet die höchste Schicht der Basis-Software. Diese Schicht enthält das Betriebssystem, das Speichermanagement, Kommunikationsservices (z.B. das Netzwerkmanagement oder Diagnoesefunktionen) und das State-Management des Steuergerätes.

Die **AUTOSAR-RTE** (Runtime Environment) bildet die oberste Kommunikationsschicht, auf der die Applikation aufbaut. Diese Schicht stellt dem Application Layer Dienste und Informationen so zur Verfügung. Die Applikation ist somit völlig unabhängig vom Steuergerät.

Über die **Complex Driver** wird der direkte Zugriff auf die Peripherie des Microcontrollers ermöglicht. Dieser direkte Zugriff ist für die Auswertung einiger Sensoren oder die Ansteuerung bestimmter Aktoren notwendig. Die Implementierung ist nicht standardisiert und erfolgt in Abhängigkeit vom Einsatzfall und der Beschaltung des Steuergerätes.

2.4.4 Definitionen

Gemäß AUTOSAR-Richtlinien gelten folgende allgemeine Definitionen der Module: [13]

Driver (Treiber):
Ein Treiber realisiert die Kontrolle und den Zugriff auf interne oder externe Baugruppen oder Controller.
Es wird zwischen zwei Arten von Treibern unterschieden

1. Interne Treiber für Baugruppen, welche im Mikrokontrollers enthalten sind. Die Treiber sind Bestandteil des Microcontroller Abstaction Layers.
2. Externe Treiber für Baugruppen, welche nicht Bestandteil des Mikrokontrollers sind und über interne Treiber in das System eingebunden werden (z.B. LCD via SPI). Diese Treiber sind Bestandteil des ECU Abstraction Layers.

Interface:
Ein Interface realisiert die Abstraktion der Hardware für die übergeordneten Ebenen. Es unterstützt eine einheitliche Schnittstelle für den Zugriff auf die spezifischen Treiber, unabhängig von der Anzahl der Baugruppen und ob diese Bestandteil des Mikrokontrollers sind oder extern angebunden werden (z.B. einheitlicher Zugriff auf internen und externen EEPROM).
Das Interface hat keinen Einfluss auf den Inhalt der Daten, welche vom oder zum Treiber übertragen werden. Das Interface ist Bestandteil des ECU Abstraction Layers.

Handler:
Ein Handler ist ein spezielles Interface, welches den konkurrierenden, mehrfachen asynchronen Zugriff auf Baugruppen kontrolliert. Er unterstützt das Puffern, die Arbitrierung und das Multiplexen von Daten.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Der Handler hat wie das Interface keinen Einfluss auf den Inhalt der Daten und ist Bestandteil des ECU Abstraction Layers.

Manager:
Ein Manager bietet spezielle Unterstützung für den Zugriff auf Baugruppen. Er wird immer dann benötigt, wenn ein Handler mit seinen Eigenschaften für den Zugriff und die Nutzung der Treiber nicht ausreichend ist.
Ein Manager kann Daten prüfen und anpassen, welche zwischen Treiber und Applikation ausgetauscht werden.
Der Manager ist Bestandteil des Service-Layers.

2.4.5 Allgemeine Architektur eines Treibers

Neben den vorgefertigten Treibern müssen in diesem Projekt diverse Treiber selbst erstellt werden. Diese Treiber sind in ihrer Architektur in mehrere Ebenen aufgeteilt.

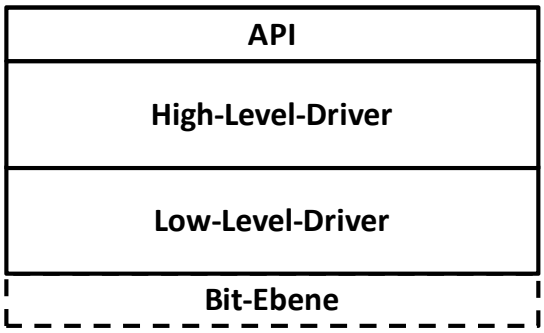


Abbildung 18: Allgemeine Treiberarchitektur

Die **Bit-Ebene** besteht rein aus Makros zum Setzen und Rücksetzen einzelner Bits in Registern oder aus Strukturen, welche den Aufbau des Registers abbilden. Diese werden für das gesamte Projekt zentral in einem Header-File definiert.

Der **Low-Level-Driver** enthält die elementaren Funktionen zur Abstraktion der Register.
Es werden z.B. folgende Funktionen realisiert.

- Übertragungsgeschwindigkeit festlegen
- Übertragungsformat festlegen
- Art der Interrupts festlegen
- Daten byteweise senden/empfangen

Durch die **High-Level-Driver** werden die Elementarfunktionen der unteren Treiber-Schicht zu komplexen Funktionen zusammengefasst.
Es werden z.B. folgende Funktionen realisiert.

- Initialisieren Controller/Register
- Datenrahmen senden

Die **API** (Application Programming Interface) bildet die zentrale Programmierschnittstelle des Treibers nach außen.
Durch die API werden z.B. die Call Back Funktion aufgerufen, welche Ereignisse an höhere Schichten melden.

In den bisherigen Entwicklungsprojekten in Zusammenarbeit mit den Automobilherstellern

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

3 Anforderungserfassung

Bevor die Spezifikation des Universalgateway vorgenommen werden kann, müssen die Anforderungen an dieses Projekt ermittelt werden. Die Anforderungen ergeben sich aus den internen Vorgaben der Entwicklungsabteilung und der Anforderungen der einzelnen Entwicklungsprojekte.

Die bei den Automobilherstellern eingesetzten Bussysteme und Protokolle unterscheiden sich zum Teil sehr stark. Ferner stellen die einzelnen Entwicklungsprojekte verschiedene Anforderungen an ein Universalgateway. Aus diesem Grund sollen die Anforderungen der Projekte übergreifend für die Automobilhersteller in einer übersichtlichen Matrix erfasst werden. Diese Matrix bildet zusammen mit den Erfordernissen aus der firmeninternen Soft- und Hardwareentwicklung die Basis für die Anforderungsanalyse und Spezifikation des Systems.

3.1 Anforderungen der Soft- und Hardwareentwicklung

Nachfolgend sind die Anforderungen der firmeninternen Hard- und Softwarentwicklung aufgeführt. Diese Anforderungen bilden die Basis für die Wahl der Prozessorplattform und der Systemarchitektur.

- Basis muss die firmenweit eingesetzte Hardwareplattform NEC V850 Fx3 sein.
- Es ist die Webasto SW-Architektur einzuhalten.
- Es muss ein OSEK/VDX kompatibles Betriebssystem eingesetzt werden.
- Zur Abstraktion des Mikrocontrollers sind AUTOSAR-konforme Layer zu verwenden.
- Funktionen und Module sollen ohne größere Anpassungen zwischen dem Universalgateway und Entwicklungsprojekten ausgetauscht werden können.
- Eine frühe Qualifizierung projektspezifischer Module soll möglich sein.

3.2 Anforderungsanalyse

In Anhang A sind die funktionalen Anforderungen der einzelnen Entwicklungsprojekte (sortiert nach Automobilhersteller) an das Universalgateway dargestellt. Die dargestellten Anforderungen sind das Ergebnis aus der Befragung der Entwicklerteams und aus den Gesprächen mit Mitarbeitern einiger Automobilhersteller sowie der Analyse aktueller und geplanter Entwicklungsprojekte.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Die Anforderungen der Entwicklungsprojekte (siehe Anhang A) wurden zusammengefasst und ausgewertet. Die an das Universalgateway gestellten Erwartungen sind sehr vielfältig. Um diese einzugrenzen, wurden in Gesprächen mit den Entwicklerteams die folgenden Anforderungen an das Universalgateway herausgearbeitet und als Basis für die Auslegung der Hardware und den Umfang der zu implementierenden Funktionalitäten definiert.

Kategorie	**Anforderung**
Einsatzumgebung:	Der Einsatz erfolgt vornehmlich im Fahrzeug
Betriebsspannungsbereich:	9 bis 17 V (eine Lkw-Variante bis 32V ist vorzusehen)
Schnittstellen:	Es sind folgende Schnittstellen vorzusehen: • 2 x CAN High-Speed (125 / 500 kbit/s) • 2 x CAN Low-Speed (100 kbit/s) als Bestückungsvariante • 2 x LIN V2.1 (9,6 / 19,2 kbit/s) • 2 x Webasto-Bus • 2 x Digital In (Low < 2 V / High maximal ≥ 7 V) System soll selektiv über digitalen Eingang weckbar sein. • 2 x Digital Out (High-Side 2,5 A) • 2 x Digital Out (Halbbrücke 2,5 A) • 2 x Analog In (0 bis 18,5 V / LKW-Version 0 bis 32 V)
Stromaufnahme:	Das System muss ruhestromfähig sein (schaltet selbst ab). • Ruhestrom < 60 µA • Weckbar über Datenbusse und selektiv über Digital In
Datenlogger:	Das System soll folgende Daten für eine spätere Auswertung aufzeichnen können: • Daten der Datenbusse • Daten der digitalen und analogen Ein- und Ausgänge • Werte und Daten interner Funktionen
Anzeige Display:	Zur Visualisierung von Daten sollen folgende Informationen auf einem Display dargestellt werden: • Daten der Datenbusse • Daten der digitalen und analogen Ein- und Ausgänge • Werte und Daten interner Funktionen Meist reicht eine einfache Darstellung der Daten.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Kategorie	Anforderung
Diagnosefunktionen:	Das System soll folgende Diagnoseprotokolle unterstützen • Webasto-Bus • KWP-2000 • UDS • XCP
Gateway-Funktionen:	Das System soll als Gateway zwischen den Bussystemen konfiguriert werden können. • Router von Botschaften (schneller Durchreicher) • Manipulation von Daten und Informationen • Erstellen und Senden eigener Botschaften
Programmierung:	Das System soll frei in C programmierbar sein. • Konfigurationsmöglichkeit zur einfachen Anpassung • vorbereitete Konfigurationen (z.B. als Datenlogger) • NEC V850 Fx3-Prozessor • OSEK als Betriebssystem • AUTOSAR-konforme Abstraktionsschichten • SW-Module aus Entwicklungsprojekten müssen austauschbar sein

Tabelle 3: Anforderungen an das Universalgateway

Innerhalb des Rahmens dieser Diplomarbeit kann nur ein Teil der ermittelten Anforderungen an ein Kfz-Universalgateway realisiert werden.
Folgende ermittelte Anforderungen werden in dieser Diplomarbeit nicht betrachtet:

- Auslegung und Erstellung der Hardware
- Funktionen zum Aufzeichnen von Daten (Datenlogger)
- LIN-Kommunikation (Treibererstellung)
- Diagnoseprotokolle KWP2000, UDS und XCP
- Konfigurierungsmöglichkeit zur einfachen Anpassung durch den User
- Fertige Konfigurationen/Programme für einzelne Anwendungsfälle erstellen

Im Anschluss an diese Diplomarbeit werden diese Anforderungen in einem weiterführenden Projekt realisiert.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

4 Systementwurf

Innerhalb des Systementwurfs werden die in Kapitel 3 erfassten Anforderungen an das System näher spezifiziert. Die verwendete Hardware wird beschrieben, die Softwarearchitektur festgelegt und die einzelnen Softwaremodule spezifiziert.

4.1 Hardware

Die Softwareentwicklung für das Universalsteuergerät wird auf dem Evaluation Board „AB-050-FX3-P“ realisiert, welches mit dem NEC V850ES FK3 Prozessor ausgestattet ist [27].
Dieser Prozessor bietet mit 176 Pins genügend Anschlussmöglichkeiten für die geforderte Peripherie und besitzt neben 1024 kByte Codeflash, 32 kByte Dataflash und 60 kByte RAM fünf integrierte CAN-Controller, sowie 8 UART-Schnittstellen. Der eingesetzte FK3-Prozessor gehört zur selben Prozessor-Familie, wie er auch in anderen Entwicklungsprojekten der Webasto-AG eingesetzt wird. Die Erstellung der Programme erfolgt über die Entwicklungsumgebung „Multi for V800“ der Firma Green Hills. Auf dem verwendeten Board ist bereits die Spannungsversorgung und die Beschaltung mit einem 16 MHz-Quarz realisiert. Auf diese Weise können schnell einfache Programme erstellt und erprobt werden. Zur Programmierung des Prozessors und zum Debuggen der Programme wird der „Minicube“ der Firma NEC eingesetzt.
Somit ist die Forderung nach einer einheitlichen Entwicklungsplattform sichergestellt.

4.1.1 Zusatzbeschaltung

Zur Realisierung der Anforderungen an die Ein- und Ausgänge sowie dem Powermanagement wurde das Starter-Board mit einer selbsterstellten Zusatzplatine erweitert. Die einzelnen Steuer- und Signalleitungen wurden über Flachbandkabel oder Einzelleitungen mit dem Starter-Board verbunden. Auf diese Weise kann die Pin-Belegung je nach Bedarf angepasst werden. Ferner wurde die Beschaltung der auf dem Starter-Board befindlichen CAN- und LIN-Transceiver an die Erfordernisse des Universalgateways angepasst.

Die für die Erweiterung des Evaluation Board genutzten Schaltungen wurden in Anlehnung an bereits existierende Steuergeräten und Schaltungen erstellt. Auf diese Weise soll sichergestellt werden, dass die im Universalgateway realisierten Funktionen im Verhalten zu Serien-Steuergeräten identisch ist.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Digitale Eingänge:

Zur Auswertung von Schaltsignalen wurde ein einfacher Spannungsteiler mit einer Spannungsbegrenzung über eine Z-Diode realisiert (Abbildung 19).

Zur Vermeidung von fehlinterpretierten Signalen durch den Prozessor bei schwankenden Spannungspegeln am Eingang dieser Schaltung muss das Signal an einen Prozessor-Pin angeschlossen werden, welcher als digitaler Eingang mit Schmitt-Trigger-Charakteristik konfiguriert werden kann. Beim Fx3-Prozessor gibt es zwei verschiedene Charakteristika (30/70% und 40/80%) je nach dem, welcher digitale Eingang genutzt wird.

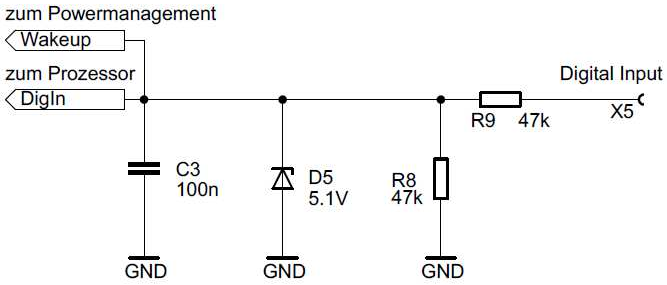


Abbildung 19: Beschaltung digitaler Eingang

Bei einer Prozessorspannung von 5V ergeben sich folgende Schaltschwellen:

- Pin mit 30/70% Charakter: Low für U_{in} < 1,8V / High für U_{in} > 4,2V
- Pin mit 40/80% Charakter:: Low für U_{in} < 2,4V / High für U_{in} > 4,8V

Zur Erfüllung der Anforderungen werden Pins mit einer 30/70% Charakteristik verwendet.

Analoge Eingänge:

Der Fx3-Prozessor kann über den integrierten AD-Wandler Spannungen von 0 V bis zur Höhe der Referenzspannung mit einer Auflösung von 10 Bit digitalisieren. Die Referenzspannung ist beim verwendeten Evaluation Board fest mit der Versorgungsspannung des Prozessors (5 V) verbunden. Das System soll gemäß Anforderungsanalyse Spannungen von 0 bis 18,5 V digitalisieren können. Aus diesem Grund wird die Signalspannung am analogen Eingang über einen Spannungsteiler aufbereitet und zur Vermeidung von Überspannungen mittels einer Z-Diode begrenzt. Durch die in Abbildung 20 dargestellte Schaltung kann das Eingangssignal mit einer Auflösung von 18 mV digitalisiert werden.

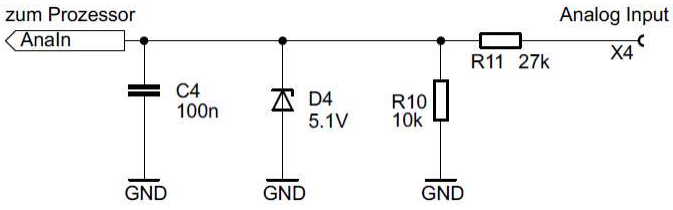


Abbildung 20: Beschaltung analoger Eingang

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Digitale Ausgänge:

Für das Schalten von Lasten wurde ein integrierter Leistungstreiber gewählt. Dieser High-Side-Switch integriert einen ESD-Schutz, Überspannungs-, Übertemperatur, Überstromschutz und kann Ströme bis 2,6 A schalten. Die in Abbildung 21 dargestellte Schaltung entspricht der Vorgabe aus dem Datenblatt.

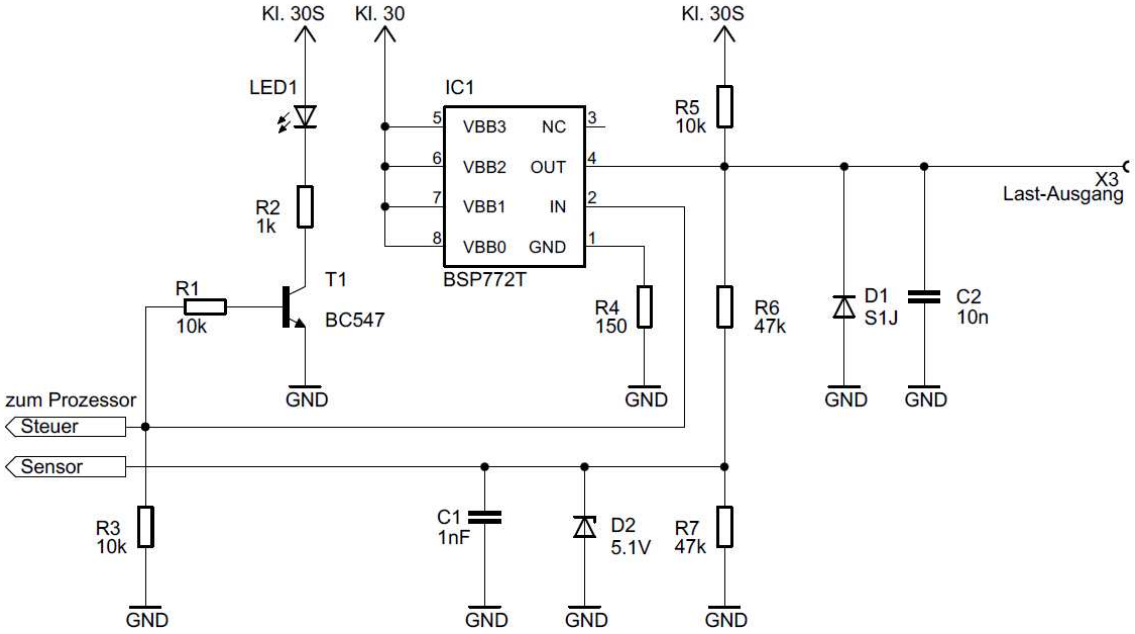


Abbildung 21: Schaltausgang

Die Realisierung einer Halbbrücke kann innerhalb der Diplomarbeit nicht realisiert werden.

Für die Erkennung von Fehlern (Kurzschluss/Unterbrechung) wird das Ausgangssignal über einen Spannungsteiler wieder durch den Prozessor zurückgelesen und mit dem Steuersignal verglichen. Es ergeben sich folgende Zustände:

Steuer-Signal	**Sensor-Signal**	**Zustand**
Low	Low	OFF (kein Fehler)
Low	High	Unterbrechung oder Kurzschluss nach U_{Batt}
High	Low	Kurzschluss nach Masse
High	High	ON (kein Fehler)

Tabelle 4: Zustände der digitalen Ausgänge

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Die Fehlererkennung unterliegt folgenden Einschränkungen:

- Ein Kurzschluss ist nur erkennbar, wenn der Ausgang angesteuert wird.
- Eine Unterbrechung ist nur erkennbar, wenn der Ausgang nicht angesteuert wird.

Zusätzlich wird über einen Transistor der Zustand der Steuerleitung durch eine LED visualisiert.

LIN-Bus/Webasto-Bus:

Auf dem Starter-Board sind die acht UART-Schnittstellen des Prozessors wahlweise über einen RS232-Pegelwandler oder über einen LIN-Transceiver nach außen gelegt. Die Beschaltung von UART 0 und UART 1 wurden so angepasst, dass diese der LIN-Beschaltung eines Master-Knotens in Kraftfahrzeugen bzw. einem Webasto-Bus-Teilnehmers entspricht.

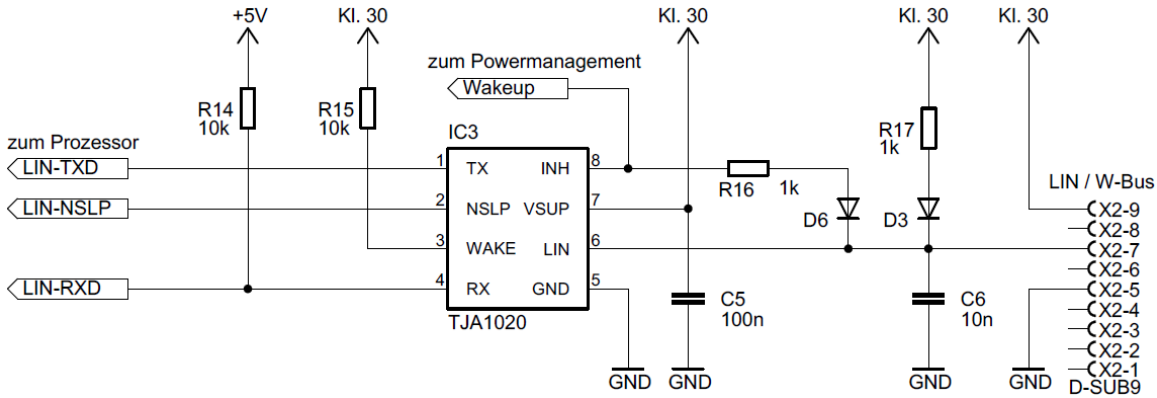


Abbildung 22: Beschaltung LIN-Bus/Webasto-Bus

CAN-Bus:

Die Ausgänge der fünf CAN-Controller des Prozessors sind über einen High-Speed-CAN-Transceiver nach außen geführt. Die Beschaltung des Evaluation Board ist dabei vorgegeben. Der CAN-Transceiver wechselt nach dem Anlegen der Versorgungsspannung automatisch in den aktiven Betriebszustand. Soll der CAN-Transceiver über den Mikrocontroller gesteuert werden, dann muss die Schaltung mindestens entsprechend Abbildung 23 angepasst werden. Die dargestellte Ausgangsbeschaltung erfüllt jedoch nicht die Anforderungen der Kraftfahrzeughersteller. Es fehlen die Gleichtaktdrossel zur Unterdrückung von Störsignalen auf dem CAN und ein zusätzlicher ESD-Schutz. Diese Komponenten können nicht direkt auf dem Board nachgerüstet werden. Die Anpassung der Schaltung und die Integration der notwendigen Komponenten erfolgt innerhalb der endgültigen Hardware.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

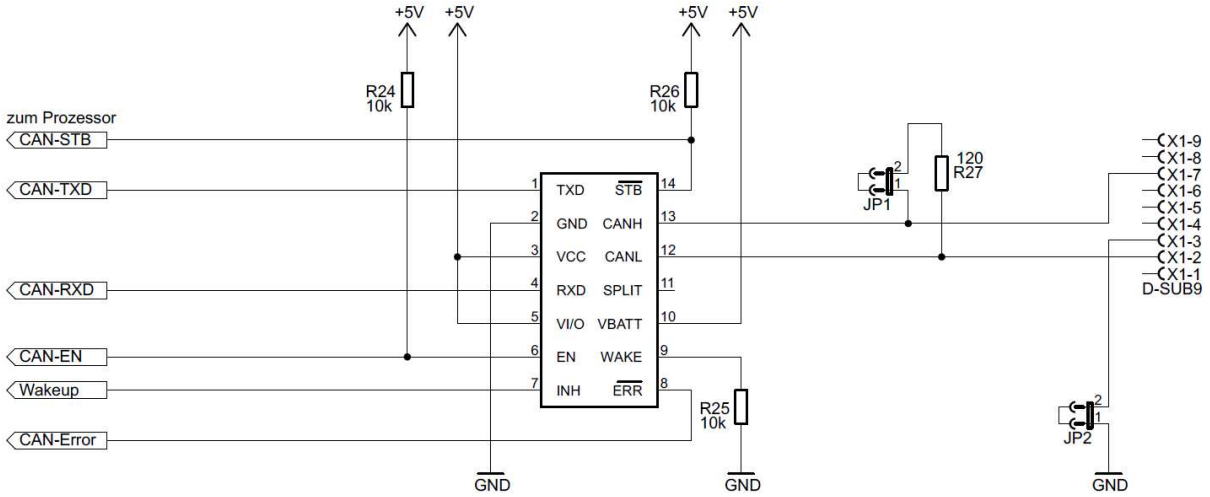


Abbildung 23: Beschaltung CAN-Bus

Display:

Zur Visualisierung von Daten und Zuständen wurde ein Grafik-LCD über eine SPI-Schnittstelle an den Prozessor angebunden. Das Display hat bei einer Größe von 2,1“ eine Auflösung von 132 x 176 Pixeln mit einer Farbtiefe von 16 Bit. Die notwendige Anpassung der Signalpegel und die Steuerung der Display-Beleuchtung ist durch den Displaylieferanten auf einer eigenen Platine realisiert. Innerhalb einer späteren Weiterentwicklung werden diese Komponenten dann direkt auf der Zielhardware integriert.

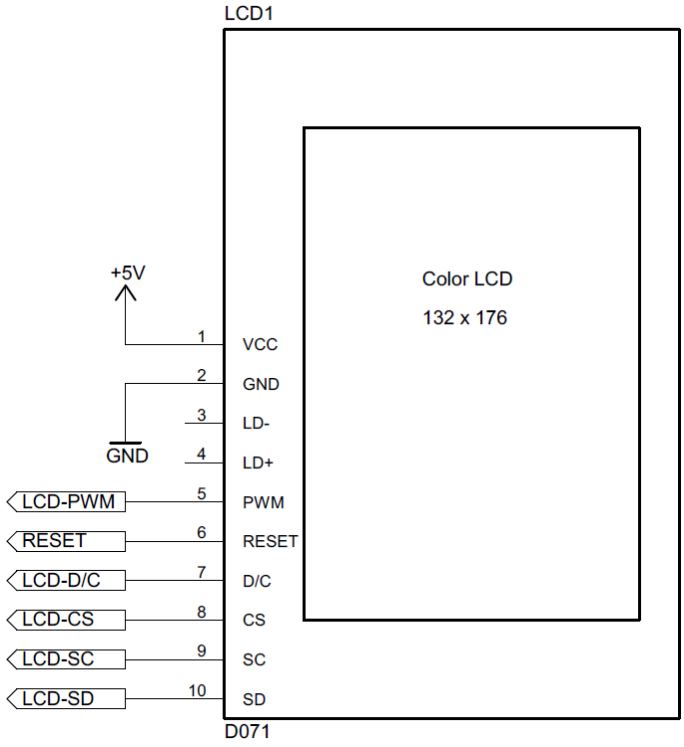


Abbildung 24: LCD-Display

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Für die Diplomarbeit wurde das Display über ein Flachbandkabel entsprechend Abbildung 24 an den Prozessor angebunden.

Powermanagement:

Damit das Kfz-Universalgateway im Fahrzeug eingesetzt werden kann, ist es erforderlich, die Ruhestromforderungen der Kraftfahrzeughersteller einzuhalten. Die Anforderungsanalyse hat hierfür einen maximalen Ruhestrom von 60µA ergeben.

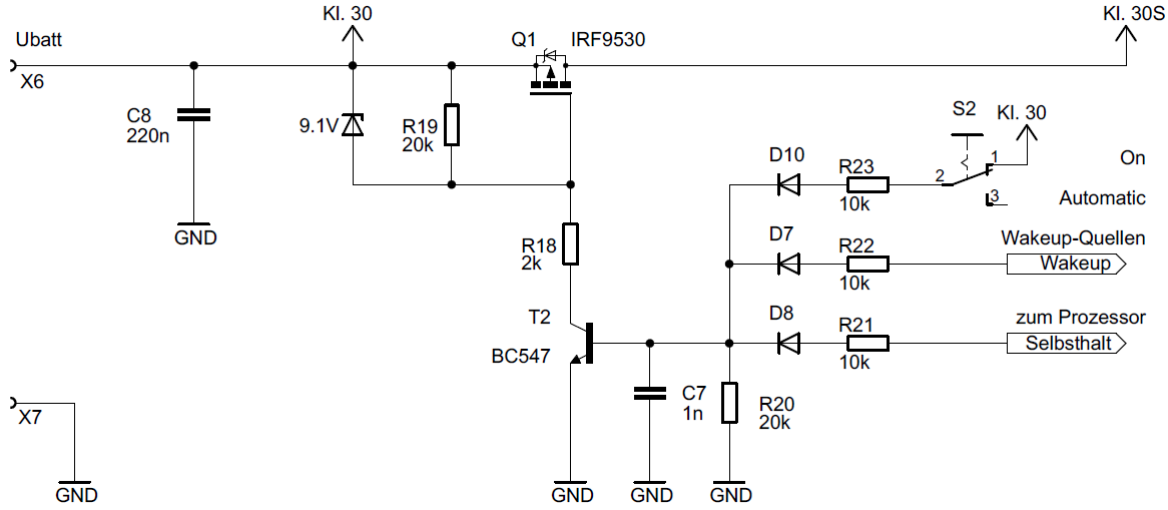


Abbildung 25: Powermanagement

Durch die in Abbildung 25 dargestellte Schaltung ist es möglich, alle Komponenten inkl. des Prozessors von der Versorgungsspannung zu trennen, wenn das System in den Sleep-Mode übergeht. Auf diese Weise kann die Anforderung an den Ruhestrom eingehalten werden.
Das Wecken des Steuergerätes erfolgt über die an der Basis des Transistors T2 angeschlossenen Wakeup-Quellen. Zum Wecken reicht ein kurzer Impuls aus. Direkt nach dem der Mikrocontroller mit Strom versorgt wird, muss dieser die Selbsthaltung übernehmen. Als Wakeup-Quellen können die digitalen Eingänge sowie die Weckausgänge der LIN und CAN-Transceiver genutzt werden. Über den Schalter S2 besteht die Möglichkeit das Universalgateway manuell aus dem Sleep-Mode zu wecken und das Abschalten des Steuergerätes zu verhindern.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

4.1.2 Ausstattung der Entwicklungsumgebung

Zur Realisierung des Kfz-Universal-Gateways steht nach den im vorangegangenen Kapitel beschriebenen Anpassungen folgende Entwicklungsumgebung zur Verfügung:

- 2 x LIN/WBus (8 Kanäle sind auf dem Starter-Board möglich)
- 2 x High-Speed CAN (5 CAN-Kanäle sind auf dem Starter-Board möglich)
- 4 x Digital Out (High-Side mit Fehlererkennung Kurzschluss/Unterbrechung)
- 2 x Digital In (Schaltpunkt ca. 5 V)
- 2 x Analog In (0…18,5 V)
- 3 x Taster (frei verwendbar z.B. zur Steuerung von Menüs)
- 2,1‘ Grafik-Display zur Anzeige von Daten/Informationen

Der Zugriff auf die digitalen und analogen Ein- und Ausgänge ist über 4 mm Laborbuchsen realisiert. Die Lin/WBus- und CAN-Schnittstellen sind über Buchsen Sub-D neunpolig zugänglich.

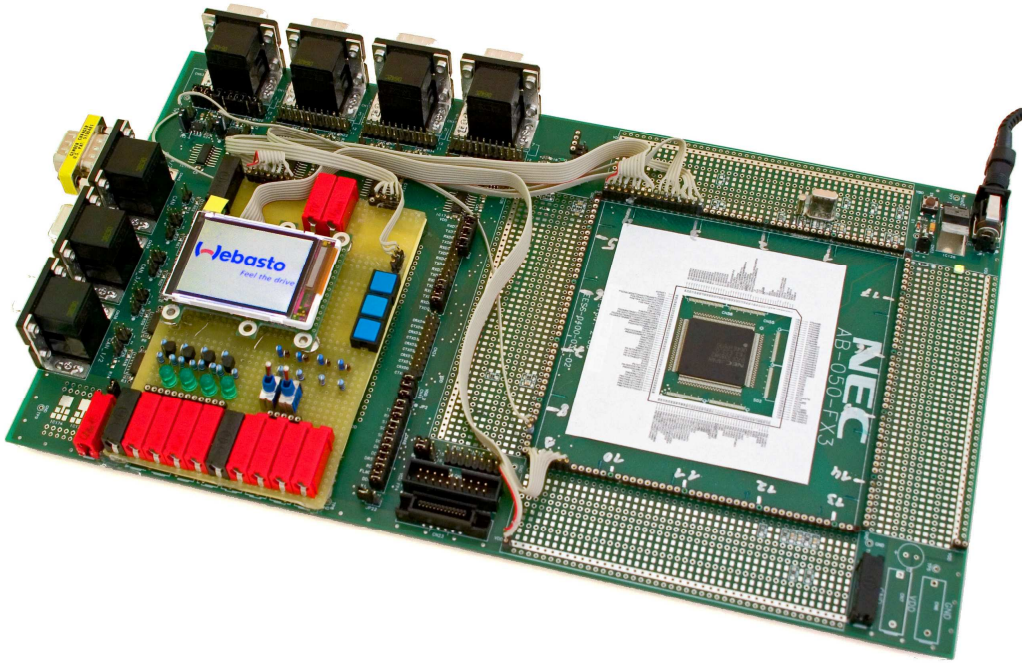


Abbildung 26: Ansicht Gesamtsystem

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

4.2 Geplante Software-Architektur

Gemäß der allgemeinen AUTOSAR-Architektur [13] sind die einzelnen Module auf verschiedene Abstraktions-Ebenen aufgeteilt. Für das Kfz-Universalgateway ergibt sich aus dieser Forderung folgende Softwarearchitektur:

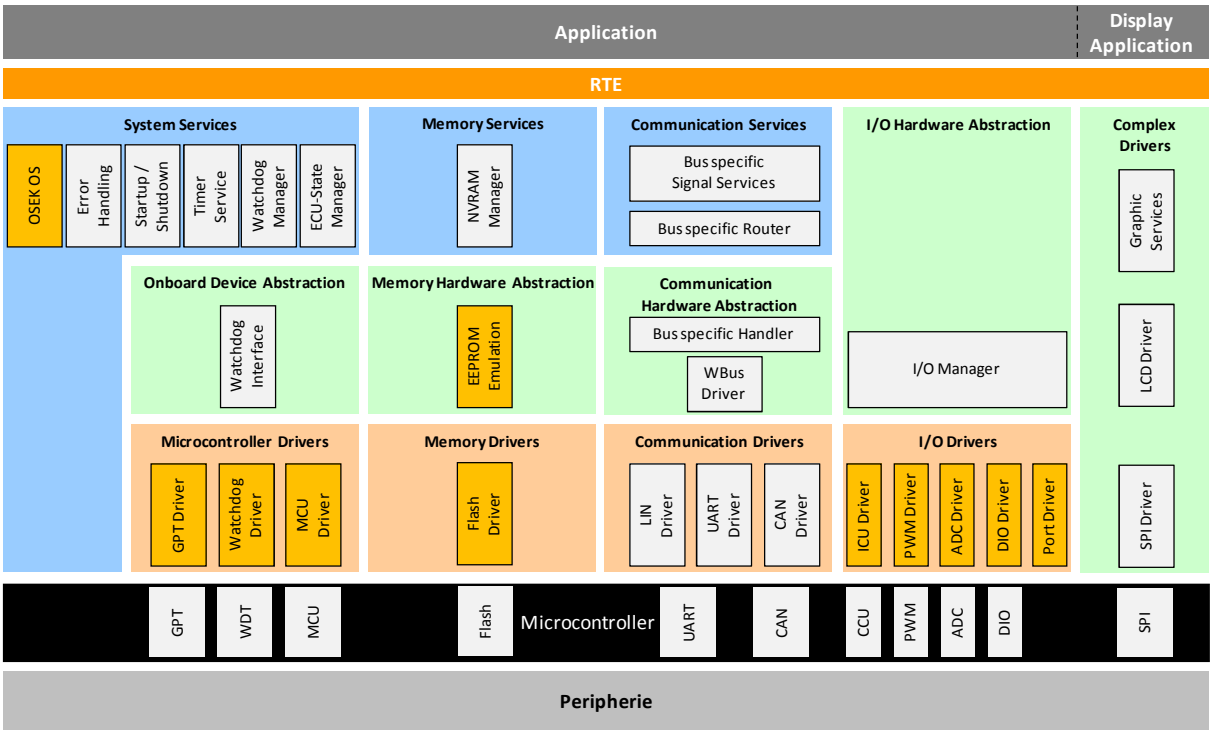


Abbildung 27: Übersicht der geplanten Softwarearchitektur

Die gelb markierten Module sind in Form einer Standard-Software vorhanden. Diese Module müssen entsprechend der funktionellen Anforderungen konfiguriert und integriert werden. Die hellgrau dargestellten Module der Basis-Software müssen selbst erstellt und integriert werden. Nachfolgend werden die einzelnen Module der Software sowie die Anforderungen und Eigenschaften spezifiziert.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

4.3 Spezifikation der Module

4.3.1 Input/Output-Signal-Abstraktion

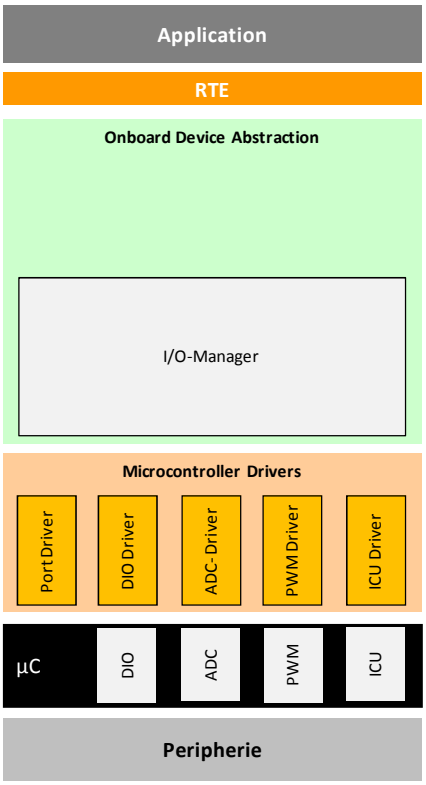


Abbildung 28: IO-Signal-Abstraktion

Das Modul „Input/Output-Signal-Abstraktion“ übernimmt innerhalb der Initialisierungsphase die Konfiguration der Ports und Pins und stellt einheitliche Services zur Ansteuerung und zum Auslesen der analogen und digitalen Ein- und Ausgänge zur Verfügung. Die Hardware des Steuergerätes wird dabei weitgehend abstrahiert.

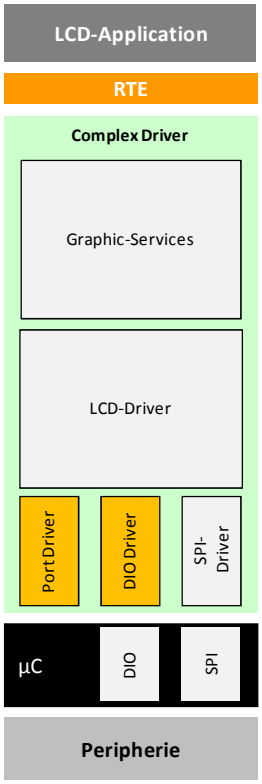
Der **I/O- Manager** übernimmt die einheitliche Kommunikation der Module der Applikation zu den verschiedenen Ein- und Ausgängen des Steuergerätes. Im Signal-Manager werden dazu die die Eingangs- und Ausgangssignale geprüft und in ihren Werten passend skaliert. Ferner wird hier das Fehlerhandling der digitalen Ausgänge realisiert. Erkennt der IO-Manager einen Kurzschluss, dann muss der betroffene Schaltausgang sofort stromlos geschaltet werden.

Die Treiber des MCAL liegen als AUTOSAR-Treiber vor. Diese sind entsprechend der Anforderungen zu konfigurieren und zu implementieren.
Der **Port-Driver** stellt Services für die Konfiguration aller On-Chip-Ports und -Pins zur Verfügung und übernimmt die Initialisierung aller Port und Pins. Ferner ist der Port-Driver für das Umschalten der Pin-Konfiguration zur Laufzeit verantwortlich. Dieser Treiber muss entsprechend der Pin-Belegung des Prozessors konfiguriert werden.
Der **DIO-Driver** stellt zentral die Services für das Auslesen und Beschreiben der als digitale Ein- oder Ausgänge konfigurierten Pins und Ports des Prozessors zur Verfügung. Für eine Abstraktion der Hardware sind symbolische Bezeichnungen für die Pins zu verwenden.
Der **ADC-Driver** initialisiert und steuert die internen Analog-Digital-Converter (ADC) und stellt Services für das Starten und Stoppen der Digitalisierung von Signalen zur Verfügung.
Für eine Abstraktion der Hardware sind symbolische Bezeichnungen für die ADC-Kanäle zu verwenden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Der **PWM-Driver** stellt Services zur Initialisierung und zum Steuern der Timer des Mikrokontrollers im PWM-Mode zur Verfügung. Es wird genutzt, um softwareunabhängig pulsweiten-modulierte Signale zu erzeugen.
Für eine Abstraktion der Hardware sind symbolische Bezeichnungen für die PWM-Kanäle zu verwenden.
Der **ICU-Driver** (ICU = Input Capture Unit) wird zum Demodulieren von PWM-Signalen, Zählen von Impulsen, Messen von Frequenzen und zur Generierung von Wakeup-Interrupts durch die Nutzung der Timer im Capture-Mode (Input Capture Unit) eingesetzt.
Für eine Abstraktion der Hardware sind symbolische Bezeichnungen für die PWM-Kanäle zu verwenden.

4.3.2 LCD-Funktionen



Über die LCD-Funktionen werden Informationen von den angeschlossenen Bussystemen und interne Informationen der Applikation grafisch dargestellt. Zur Entlastung der eigentlichen Applikation erfolgt die Aufbereitung und Darstellung der Informationen durch eine Separate LCD-Applikation.

In den **Graphic-Services** werden die Grundfunktionen des LCD-Treibers zu komplexen Funktionen zusammengefasst und stellen diese über einheitliche Services der Applikation zur Verfügung, so dass beispielsweise komprimierte Grafiken durch einen einfachen Funktionsaufruf dargestellt werden können.
Der **LCD-Driver** übernimmt die Anbindung des LCDs via SPI-Schnittstelle an das System und stellt Grundfunktionen für Initialisierung und die graphische Darstellung (z.B. Window, Point, clear screen) zur Verfügung.

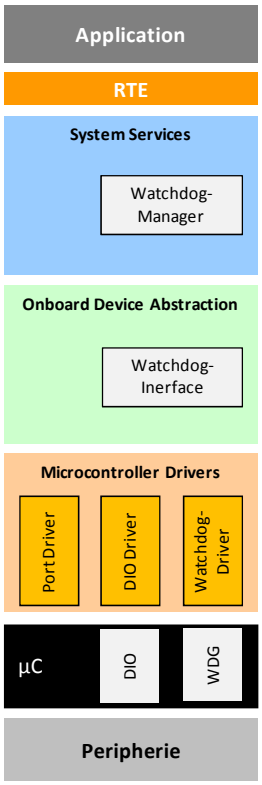
Abbildung 29: LCD-Funktionen

Der **SPI-Driver** stellt die Verbindung zwischen Prozessor und LCD her. Da sich die Kommunikation zum LCD auf das Versenden von 16-Bit-Werten beschränkt und große Datenmengen in kurzer Zeit zu übertragen sind, kommt hier ein selbst erstellter Treiber zum Einsatz. Der AUTOSAR-Treiber wäre für diesen Einsatz zu aufwendig.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Zum Beschreiben der kompletten Displayfläche müssen die Werte für 23232 Pixel mit je 2Byte übertragen werden. Bei einer softwaregesteuerten Datenübertragung wird hierfür sehr viel Zeit benötigt, in der die Aufgaben der eigentlichen Applikation nicht bearbeitet werden können. Aus diesem Grund wird die Kommunikation zum LCD asynchron gestaltet und wo immer möglich der **DMA-Mode** des Prozessors genutzt.
Die Steuerleitungen des LCD werden über den **DIO-Driver** angesprochen. Über den **Port-Driver** erfolgt die Initialisierung der Port-Pins im geforderten Betriebs-Mode. Ferner wird die SPI-Schnittstelle für die Initialisierungsphase des LCD auf den Port-Mode umgeschaltet.

4.3.3 Watchdog



Die Aufgabe des Watchdogs ist es, bei einem erkannten Systemfehler eine Neuinitialisierung (Reset) des Systems einzuleiten. Diese Funktion wird sowohl über den prozessorinternen Watchdog als auch über eine externe Ressource (z.B. innerhalb des Spannungsreglers) realisiert. Durch die Verwendung eines internen und externen Wachtdog kann ein abgestuftes Verhalten im Fehlerfall realisiert werden.

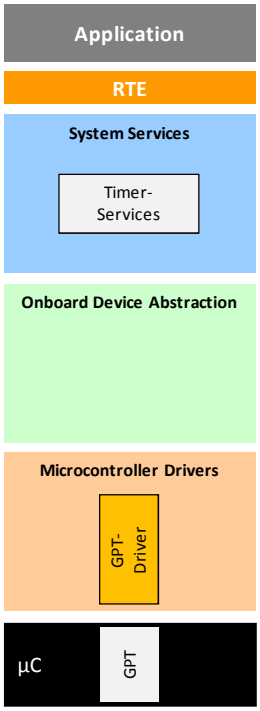
Der **Watchdog-Manager** übernimmt hierbei die Aufgabe, unabhängig von der Applikation den internen und externen Watchdog zu initialisieren und regelmäßig zu triggern. Zur Erfüllung dieser Aufgabe benötigt der Watchdog-Manager einen eigenen zyklischen, nicht präemptiven Task mit hoher Priorität.
Das **Watchdog-Interface** bildet eine einheitliche AUTOSAR-Schnittstelle und fasst die unterschiedlichen Funktionen zur Initialisierung und Steuerung des internen und externen Watchdogs zu einheitlichen Services zusammen (Hardware-Abstraktion).

Abbildung 30: Watchdog

Der **Watchdog-Driver** stellt dem System Services zur Initialisierung, zum Anpassen und Triggern des externen und mikrokontrollerinternen Watchdog zur Verfügung. Ferner wird hier das Verhalten des Systems beim Überlauf des internen Watchdog festgelegt.
Die Ansteuerung des externen Watchdogs (auf der Test-Hardware nicht vorhanden) erfolgt durch das zyklische Toggeln eines Prozessor-Pins über den **DIO-Driver**. Dieser Pin wird entsprechend im **Port-Driver** konfiguriert.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

4.3.4 Timer-Services

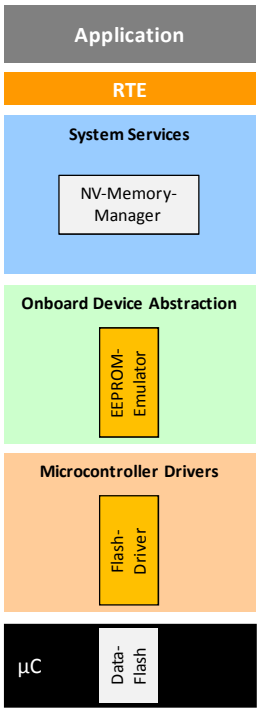


Über die **Timer-Services** wird den Modulen der Basis-Software unabhängig vom Betriebssystem die Möglichkeit zur Realisierung von Timer-Funktionen zur Verfügung gestellt.
Eine solche Timer-Funktion wird immer dann genutzt, wenn ein OS-Alarm zu aufwendig ist (z.B. zur Schrittmotor-Steuerung).
Eine Nutzung der Timer-Sevices durch die Applikation ist nicht erlaubt. Diese nutzt über die RTE ausschließlich die Möglichkeiten des OS zur Realisierung von Timer-Funktionen.

Der **GPT-Driver** (GPT = general purpose timer) stellt Services zur Konfiguration und Steuerung der mikrokontrollerinternen Timer zur Verfügung. Er unterstützt die Möglichkeit zur Nutzung von Call-Back-Funktionen und Methoden zum Scheduling. Dieser Treiber steht als AUTOSAR-Modul zur Verfügung und wird entsprechend der Anforderungen der Basis-Software konfiguriert und ins System eingebunden.

Abbildung 31: Timer-Services

4.3.5 NV-Memory



Im NV-Memory (Non-Volatile Memory) werden veränderliche Daten und Konfigurationsparameter persistent abgespeichert.
Durch die Nutzung von veränderlichen Konfigurationsparametern kann die Applikation im Verhalten angepasst werden, ohne dass die Software neu kompiliert und auf den Prozessor aufgespielt werden muss. Ferner können hier Daten zwischengespeichert werden, welche beim nächsten Start des Systems benötigt werden.

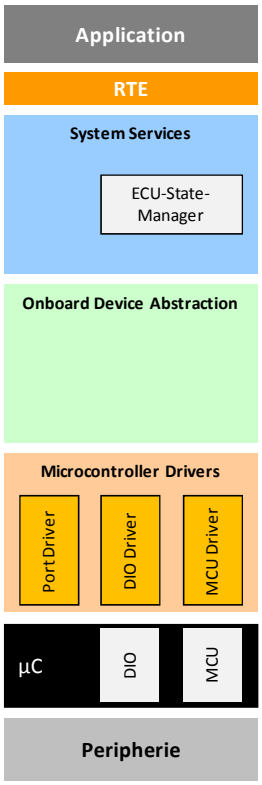
Der **NV-Memory-Manager** regelt den konkurrierenden und asynchronen Zugriff auf die Daten. Er fügt beim Auslesen von Daten aus dem Speicher diese entsprechend der gestellten Anforderungen zusammen oder zerlegt die Daten beim Schreiben in die benötigten Formate und Größen des Speichers.

Abbildung 32: NV-Memory

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Der in diesem Projekt eingesetzte Fx3-Prozessor besitzt zum persistenten Abspeichern von Daten keinen EEPROM. Für die Speicherung solcher Daten steht ein eigener Data-Flash zur Verfügung. Für den einfachen Zugriff auf diese Daten wird der **EEPROM-Emulator** genutzt. Er abstrahiert den mikrokontrollerspezifischen Adressraum und überwacht die Schreibzyklen auf den Flash. Auf diese Weise wird virtuell eine unbegrenzte Anzahl von Schreibzugriffen auf den Flash ermöglicht. Der **Flash-Driver** stellt die Grundfunktionen zum Lesen, Löschen und Schreiben auf den Flash-Speicher zur Verfügung. Er bietet weiterhin die Möglichkeit zum Setzen und Rücksetzen der Schreib- und Lösch-Flags. Außerhalb des Bootloader-Mode darf der Flash-Driver nicht zum Schreiben von Programmcode verwendet werden. Der EEPROM-Emulator und der Flash-Driver stehen als AUTOSAR-Module zur Verfügung. Diese werden entsprechend der Anforderungen der Basis-Software und der Applikation konfiguriert und in das System eingebunden.
Auf Grund der engen Zeitschiene wird dieses Modul im Anschluss an diese Diplomarbeit in einem weiterführenden Projekt realisiert.

4.3.6 ECU-Management



Über das ECU-Management wird der Zustand des Universal-Gateways und des Prozessors gesteuert. Hier wird auch das Powermanagement des Systems (z.B. durch das Trennen der Peripherie von der Versorgungsspannung) realisiert.

Der **ECU-State-Manager** kontrolliert und steuert die Zustände des Systems und des Powermanagements. Dazu wird ein eigener Zustandsautomat realisiert. Ferner steuert der ECU-Manager die Initialisierung der MCU.

Über den **MCU-Driver** werden Services für die Grundinitialisierung, die Anpassung der Konfiguration im laufenden Betrieb sowie für die Power-Down-Funktion der MCU zur Verfügung gestellt.

Dieser Treiber initialisiert die MCU-Clock, den PLL-clock-prescaller, den RAM-Bereich und ermittelt beim Eintreten eines Resets dessen Ursache.

Abbildung 33: ECU-Management

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Über den **DIO-Driver** wird die Steuerleitung für das Powermanagement bedient. Dieser Pin realisiert die Selbsthaltung, nachdem das System über ein Ereignis geweckt wurde. Aus diesem Grund muss der Prozessor-Pin innerhalb des DIO-Drivers und des **PORT-Drivers** so konfiguriert werden. dass er direkt mit der Initialisierung einen High-Pegel erhält.
Wird die Steuerleitung durch den ECU-State-Manager auf Low gelegt, erfolgt die Abtrennung der Peripherie und des Prozessor von der Versorgungsspannung (Low-Power-Mode).

4.3.7 CAN-Services

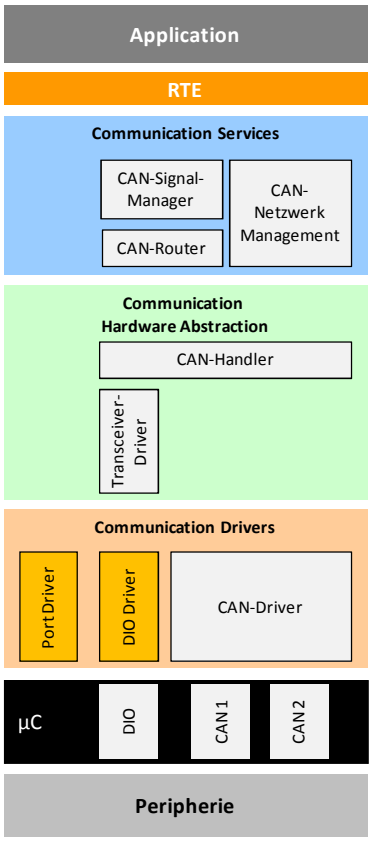


Abbildung 34: CAN-Bus:

Die CAN-Services ermöglichen die Kommunikation des Universalgateways über den CAN-Bus. Ferner wird das Routing von ganzen Botschaften zwischen den beiden CAN-Schnittstellen ohne eine Beteiligung der Applikation realisiert.

Der CAN-Router prüft an Hand einer Routingtabelle auf welchen Bus die empfangene CAN-Botschaft versandt werden soll. Ist kein Zielbus für diese Botschaft definiert, dann reicht der Router die Botschaft an den Signal-Manager weiter.
Der **CAN-Signal-Manager** bereitet diese Signale für die Weitergabe der Informationen an die Applikation auf. Dabei werden die Rohwerte passend skaliert und auf Gültigkeit geprüft. Ferner bereitet der CAN-Signal-Manager Informationen, welche auf dem CAN versandt werden sollen, entsprechend der Signalfestlegung auf. Bei der Aktualisierung von Informationen vom CAN oder im Fehlerfall wird die Applikation über den Aufruf einer Call-Back-Funktion informiert.

Der **CAN-NM-Manager** überwacht die Kommunikation mit anderen Teilnehmern im Fahrzeugnetzwerk. Ist innerhalb des Netzwerkes ein aktives Netzwerkmanagement realisiert, stößt der NM-Manager auch den gemeinsamen Übergang aller Busteilnehmer in die Busruhe an bzw. signalisiert das Wecken bei entsprechenden Ereignissen.
Das **CAN-Bus-Interface** realisiert den einheitlichen Zugriff auf die CAN-Treiber. Dabei ist es nicht von Belang, ob der CAN-Controller Bestandteil des Mikroprozessors ist oder extern zur Verfügung steht. Beim Universalgateway werden nur die internen CAN-Controller eingesetzt.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Der **CAN-Bus-Handler** steuert den Datenfluss auf dem CAN und steuert das Fehlerhandling. Ferner werden hier die CAN-Botschaften für das Netzwerkmanagement herausgefiltert.
Die Services zum Initialisieren, Reset und Anpassen der Konfiguration des CAN-Controllers werden im **CAN-Bus-Driver** umgesetzt. Ferner werden durch den CAN-Bus-Driver die Daten versandt bzw. empfangen.
Für die Steuerung des CAN-Transceivers steht ein eigener Treiber zur Verfügung. Der **Transceiver-Driver** stellt dazu entsprechende Services zur Verfügung. Des Weiteren wertet der Transeiver-Driver den Error-Pin des CAN-Transceivers zur Fehlererkennung aus.
Die Steuerung des CAN-Transceivers erfolgt dabei über den **DIO-Driver**. Alle für die CAN-Kommunikation benutzten Pins werden dabei über den PORT-Driver konfiguriert und initialisiert.

4.3.8 Webasto-Bus

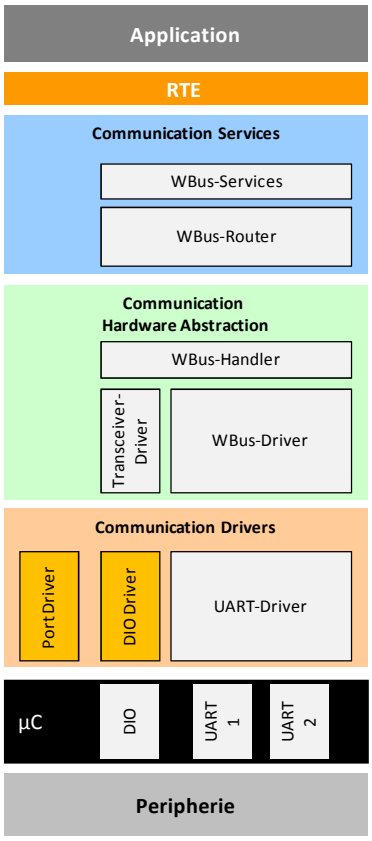


Abbildung 35: Webasto-Bus

Das Universalgateway kommuniziert mit anderen Steuergeräten über den Webasto-eigenen Bus.
Über den WBus-Router wird dabei das Routing von Telegrammen unabhängig von der Applikation realisiert.

Der **WBus-Signal-Manager** bereitet die empfangenen Informationen für die Applikation auf. Dabei werden die Rohwerte passend skaliert und auf Gültigkeit geprüft. Informationen zum Senden auf den WBus werden entsprechend der Festlegungen des WBus-Protokolls aufbereitet und an die unteren Schichten weitergegeben.
Innerhalb des **WBus-Routers** werden Datentelegramme entsprechend einer festgelegten Routing-Tabelle an andere W-Busse weitergeleitet. Durch diese Funktion ist es möglich, Geräte mit Diagnose-Tools anzusprechen, welche an einem anderen WBus angeschlossen sind, oder es können Systeme unterschiedlicher Protokollversionen eingesetzt werden.

Das **WBus-Interface** stellt Services für den einheitlichen Zugriff auf die verschiedenen WBusse zur Verfügung und steuert den Transceiver-Driver an. Das Interface realisiert auch das Starten und Abschalten der WBusse.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Dieser **WBus-Handler** steuert den Datenfluss auf dem WBus, übernimmt die Arbitrierung und löst Kollisionen auf. Ferner wird hier die Sendewiederholung bei fehlenden Antworten eines angesprochenen Teilnehmers und das Fehlerhandling realisiert.
Die Services zum Initialisieren und Anpassen der WBus-Parameter werden durch den **WBus-Driver** zur Verfügung gestellt. Er übergibt die zu sendende Daten-Telegramme byteweise an den UART-Driver. Ferner fügt er die byteweise empfangenen Werte zu Telegrammen zusammen und übergibt diese an die nächsthöhere Ebene.
Der **UART-Driver** realisiert den Zugriff auf die seriellen Schnittstellen des Mikrokontrollers. Er stellt Services zum Initialisieren und Anpassen der UART-Konfiguration sowie zum Senden und Empfangen von Daten-Bytes zur Verfügung.
Für die Steuerung des LIN-Transceivers stellt der **Transceiver-Driver** entsprechende Services zur Verfügung.
Die Steuerung des LIN-Transceivers erfolgt dabei über den **DIO-Driver**. Alle für die WBus-Kommunikation benutzten Pins werden dabei über den PORT-Driver konfiguriert und initialisiert.

4.3.9 RTE

Die RTE (Runtime Environment) stellt als höchste Abstraktionsschicht die zentrale Verbindung zwischen der Applikation und der Basis-Software her.

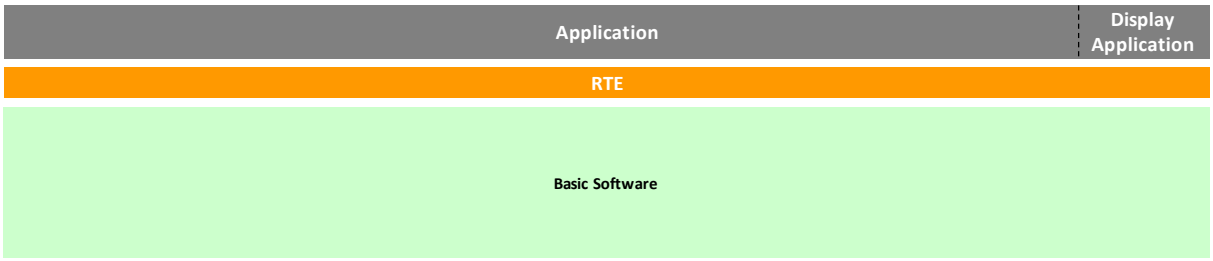


Abbildung 36: Runtime Environment

Die RTE ist stark von der Art der realisierten Applikation abhängig und ist konfigurierbar gestaltet. Der Austausch von Signalen und Informationen zwischen der Applikation und der Basis-Software erfolgt über logische Signale, welche innerhalb der RTE frei konfigurierbar sind. Durch diese Konfigurierbarkeit können Softwaremodule anderer Entwicklungsprojekte mit geringer Anpassung im Universalgateway eingesetzt werden (z.B. zur Erprobung einer Ventilsteuerung).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5 Implementierung

Im Kapitel 5 dieser Diplomarbeit wird die Implementierung der in den vorangegangenen Kapiteln ermittelten Anforderungen dargestellt.

5.1 Programmierrichtlinien

Für den Entwurf und die Implementierung von Software gelten neben den Syntax-Regeln der Programmiersprache auch formale Richtlinien, welche zur Standardisierung und Vereinheitlichung von Quelltexten beträgt. Das bekannteste Regelwerk in der Automobilindustrie ist MISRA (Motor Industry Software Reliability Association).
Die Programmierrichtlinien helfen Fehler zu vermeiden und einen gut wartbaren Code zu erzeugen. Ferner wird durch das einheitliche Erscheinungsbild die Einarbeitung in fremden Quellcode erleichtert.
Nachfolgend sind die wichtigsten Regeln, welche in diesem Projekt zu beachten waren dargestellt. [29]

- Symbole bzw. Konstanten einsetzen, wo immer es sinnvoll ist.
- Einsatz defensiver Programmierung.
- Durchsetzen eines Schichtenmodells (Strukturierung durch Abstraktion).
- Funktionen sollen zu Modulen zusammengefasst werden (Modularität).
- Der Callgraph innerhalb einer ISR ist möglichst klein zu halten.
- Einsatz von kooperativem Multitasking (Einsatz nicht unterbrechbarer Tasks).
- Globale Variablen dürfen nur an einer Stelle beschrieben werden.
- Pointer Arithmetik ist untersagt, es sei denn, der Pointer bezieht sich auf ein Array.
- Rekursive Funktionsaufrufe sind untersagt.
- Jede Funktion besitzt einen einzigen Rücksprung als letzte Anweisung.
- Die folgenden Ausdrücke sind untersagt: goto, continue, break (in Schleifen)

Neben den genannten Richtlinien wurden Namenskonventionen für die einheitliche Benennung von Objekten definiert (für Funktionen, Konstanten und Variablen etc.). [20]

- Namen sollen für sich sprechen (ähnliche Objekte besitzen ähnlich Namen).
- Aus den Namen soll die Zugehörigkeit zum Softwaremodul ergeben.
- Aus den Namen soll die Art des Objektes hervorgehen (z.B. durch Präfix oder Suffix).

5.2 Strukturierung der Softwaremodule

Der Source-Code der Software wurden entsprechend ihrer Funktion zu Modulen zusammengefasst und in einer Ordnerstruktur entsprechend der in 4.2 definierten Softwarearchitektur abgelegt. Innerhalb der Entwicklungsumgebung wurde durch die Nutzung von Subprojekten die Projektstruktur innerhalb der Entwicklungsumgebung entsprechend der der Softwarearchitektur abgebildet. Dateien, welche unmittelbar vom eingesetzten Prozessorderivat abhängig sind (z.B. Startup-File, Linker-File, Register-Deklaration oder Interrupt-Tabellen) sind im Verzeichnis „Target“ abgelegt. Über ein Header-File (target.h) wurden diese Dateien für die Einbindung in den Code zusammengefasst. Auf diese Weise wurde ein einfacher Wechsel des Prozessorderivates ermöglicht. Daten zur Dokumentation des Projektes wie z.B. die Pin-Konfiguration oder die Software-Architektur wurden im Verzeichnis „Doc“ abgelegt.

5.3 Betriebssystem

Das OSEK wird als Betriebssystem unmittelbar nach der Initialisierung der Taktsteuerung für den Prozessor und der Initialisierung des Port-Treibers gestartet und bildet die Basis für die Software des Universalgateways. Über das Betriebssystem werden die einzelnen Tasks des Projektes gesteuert und überwacht sowie die Alarme verwaltet. Das OSEK als Modul des System-Layers der Basic Software wurde innerhalb der Entwicklungsphase statisch dimensioniert und musste entsprechend der in Kapitel 4.3 ermittelten Anforderungen und des OSEK Design Guide [10] ausgelegt werden.

5.3.1 Anforderungen an die Konfiguration

Die Konfiguration des Betriebssystems soll einen möglichst effizienten und speichersparenden Code ergeben. Die Anforderungen an eine solche Auslegung sind ausführlich im OSEK Design Guide [10] beschrieben. Nachfolgend sind die wichtigsten Punkte kurz dargestellt:

- Die Aufgaben des Systems sind in einzelne Task aufzuteilen.
    - Ein Task besitzt eine Zeitachse/Time Management für sich selbst.
    - Es werden physikalische Werte verarbeitet (z.B. Temperaturkontrolle).
    - Es besteht ein Zugang zu periodischen Baugruppen (z.B. AD-Wandler) oder Gruppen von Modulen (z.B. mehrere CAN-Busse).
- Festlegen der Prioritäten:
    - Wie hoch darf die maximale Latenzzeit für eine Taskaktivierung sein?
    - Wie lange darf es dauern, bis das Ergebnis des Task zur Verfügung steht?

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

- Wie hoch ist die Laufzeit eines Task?
- Wie wird ein Task aktiviert (zyklisch oder durch ein Ereignis)?

- Auswahl der Konformitätsklasse (siehe Tabelle 5):
  - Ist es effizienter, einen wartenden Task zu aktivieren oder einen Basic-Task zu starten? (Werden nur nicht präemptive Basic-Tasks verwendet, können die sich einen Stackbereich teilen.)
  - Ist es notwendig Ressourcen für den Schutz von Daten zu nutzen oder gibt es einfachere Mechanismen? (Werden nur nicht präemptive Tasks verwendet, kann es keine Konflickte beim synchronen Zugriff auf Recourcen geben).
  - Sind zwingend Tasks mit identischer Priorität oder das mehrfache Aktivieren derselben Task notwendig?

		Zustand "Wait" zulässig?	
		Nein: Nur Basic Task	Ja: Basic und Extended Task
Mehrere Task je Prioritätsstufe und/oder mehrere gleichzeitige Aktivierungen derselben Task	*Nein*	**BCC1** Basic Conformance Class 1	**ECC1** Extended Conformance Class 1
	Ja	**BCC2** Basic Conformance Class 2	**ECC1** Extended Conformance Class 1

Tabelle 5: OSEK Konformitätsklassen

- Definieren der Scheduling-Strategie:
  - **nicht präemptiv**: Die Latenzzeit eines Task entspricht der Laufzeit des Task mit der größten Laufzeit. Die Daten, welche innerhalb des aktiven Task verarbeitet werden, sind immer konsistent. Der aktive Task kann nicht von anderen Task unterbrochen werden.
  - **präemptiv**: Die Latenzzeit ist von der Höhe der Priorität abhängig. Ein Task mit höherer Priorität verdrängt einen aktiven Task mit geringerer Priorität. Es sind jedoch Maßnahmen erforderlich, um Daten konsistent zu halten und den Zugriff auf begrenzte Ressourcen zu schützen.
  - **gemischte Systeme**: In solchen Systemen sind sowohl präemptive als auch nicht präemptive Task vorhanden. Tasks mit kurzen Laufzeiten werden als nicht präemptiv mit hoher Priorität und Tasks mit sehr langen Laufzeiten werden als präemptiv mit geringer Priorität definiert.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Ein OSEK, das als Konformitätsklasse BCC1 konfiguriert wurde und ausschließlich nicht präemptiven Tasks beinhaltet, benötigt deutlich weniger Speicher und arbeitet effizienter als Betriebssysteme, in welchen alle Funktionalitäten ausgenutzt werden (Konformitätsklasse ECC2 mit präemptiven Tasks).

5.3.2 Im Projekt ermittelte Tasks

Entsprechend der im vorangegangenen Kapitel dargestellten Kriterien wurden die benötigten Tasks für die Konfiguration des Betriebssystems ermittelt und das Betriebssystem entsprechend konfiguriert.

Der Watchdog wird als eigenständiges Überwachungssystem betrachtet. Der entsprechende Task muss automatisch starten, um bereits die Initialisierung der Softwaremodule (gesteuert durch den ECU-Manager) aktiv zu überwachen. Für den Watchdog wurde die höchste Priorität gewählt, so dass ein Verzögern der Impulse für den internen und externen Watchdog durch die Bearbeitung anderer gleichzeitig auf die Bearbeitung wartende Tasks sicher ausgeschlossen werden kann.
Neben dem Watchdog startet der Task für das ECU-Management automatisch. Alle weiteren Tasks werden dann kontrolliert durch den ECU-Manager gestartet, wenn die Initialisierung aller Softwaremodule abgeschlossen ist.
Das LCD benötigt sehr viel Zeit (fast eine Sekunde) und ein genaues Timing für die Initialisierungssequenz. Aus diesem Grund wurde für die Initialisierungphase ein eigener Task mit hoher Priorität vorgesehen.
Des Weiteren wurden die Tasks für die Kommunikation über die Bussysteme als zeitkritisch identifiziert. Daher wurde für diese im Verhältnis zu den anderen Modulen der Basic-Software eine hohe Priorität gewählt.
Die Werte der digitalen und analogen Eingänge des Universalgateway sollen zyklisch und von der Applikation unabhängig ermittelt, gefiltert und entprellt werden. Ferner muss die RTE (Laufzeitumgebung) als Schnittstelle zwischen der Basic-Software und der Applikation eigenständig Aufgaben wie z.B. die zyklische Abfrage von Daten über Bussysteme, übernehmen. Zur Realisierung dieser Anforderungen wurde je ein eigener Task für den IO-Manager und die RTE vorgesehen.
Für die eigentliche Applikation des Universalgateways wurde ein Task mit geringer Priorität vorgesehen, da hier keine zeitkritischen Aktivitäten auszuführen sind.
Alle bisherigen Tasks wurden als Basic-Tasks mit nicht präemptivem Verhalten (d.h. kein Task kann einen anderen Task verdrängen und kein Task kann auf ein äußeres Ereignis

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

warten) konfiguriert. Auf diese Weise kann es zu keinen Konflikten beim Zugriff auf begrenzte Ressourcen kommen. Diese Konfiguration ermöglicht ferner, dass ein gemeinsamer Stackbereich von diesen Tasks genutzt werden kann, ohne dass es zu Konflikten kommt.
Als einziger Task weicht der LCD-Task von diesem Vorgehen ab und wurde als präemptiver Task mit geringer Priorität ausgelegt (d.h. der LCD-Task kann von allen anderen Prozessen verdrängt werden). Dies war notwendig, da die Datenübertragung und Darstellung der Informationen auf dem LCD zum Teil sehr lange dauert (> 100ms) und somit die Abläufe der anderen Tasks sehr stark beeinflusst würden. Als präeemptiver Task mit der geringsten Priorität erfolgt die Übertragung der Informationen nun in der Zeit, in der kein anderer Task aktiv ist (der Prozess läuft somit im Hintergrund).
In der nachfolgenden Tabelle sind alle identifizierten Tasks als Übersicht dargestellt.

Modul	**Task-Bezeichnung**	**Task-Typ**	**Prio**
Watchdog	T_WDOG	autostart-non preemptive Basic-Task	200
LCD-Initialisierung	T_LCD_INIT	non preemptive Basic-Task	100
CAN	T_CAN_HANDLER	non preemptive Basic-Task	75
WBus	T_WBUS_HANDLER	non preemptive Basic-Task	70
ECU-Management	T_ECU_MANAGER	autostart-non preemptive Basic-Task	60
Input / Output	T_IO_MANAGER	non preemptive Basic-Task	55
RTE	T_RTE	non preemptive Basic-Task	50
Application	T_APPL	non preemptive Basic-Task	45
LCD-Applikation	T_LCD	full preemptive Basic-Task	40

Tabelle 6: Ermittelte Tasks

Das Anlegen und Ändern der OSEK-Konfiguration ist über einen Dongle des Herstellers geschützt. Aus diesem Grund wurde neben den in Tabelle 6: Ermittelte Tasks aufgelisteten Tasks noch drei weitere Tasks (T_10ms, T_25ms und T_100ms) inklusive der dazugehörigen Alarme konfiguriert, um innerhalb der Entwicklungsphase flexibel auf Anforderungen reagieren oder um temporär einzelne Prozesse in eigene Tasks auslagern zu können. Der Stack wurde entsprechend der zu bearbeitenden Aufgaben für alle Tasks mit großzügigen Reserven dimensioniert (mit z.B. 60 Byte für den Watchdog und 500 Byte für den LCD-Task). Wenn die Softwareentwicklung des Projektes in die Optimierungsphase übergeht, dann muss der real benötigte Bedarf an Speicher für den Stack ermittelt und die Konfiguration des OS entsprechend angepasst werden.

Die ermittelte Konfiguration des Betriebssystems entspricht der einfachsten und effektivsten Konformitätsklasse BCC1 für die Implementierung.

5.4 AUTOSAR-Treiber

Die gesamte Kommunikation der Software mit dem Mikrocontroller erfolgt über die Treiber des Microcontroller-Abstraction-Layers. Folgende Treiber standen dazu als AUTOSAR 2.1-konforme Module zur Verfügung:

- MCU-Treiber
- Watchdog-Treiber
- Port-Treiber
- DIO-Treiber
- ADC-Treiber
- GPT-Treiber
- PWM-Treiber
- ICU-Treiber
- FEE-Treiber (nicht eingesetzt)

Basis für alle Treiber bilden die ECU Configuration Parameter Definition Files, welche für jeden Treiber in Form einer XML-Date vorliegen. Die im Projekt benötigten Treiber werden zusammen in das Konfigurations-Tool geladen. Über die in Abbildung 38 dargestellte Oberfläche können dann die erforderlichen Konfigurationen vorgenommen werden.
Alle Einstellungen, welche innerhalb des Konfigurations-Tools vorgenommen werden, speichert dieses in einem zentralen File ab (*.one). Das Konfigurations-Tool erzeugt aus diesem dann AUTOSAR-konforme ECU Configuration Description Files im XML-Format, welche als Eingangsinformationen für die Code-Generatoren der einzelnen Treiber dient. Die Erzeugung des Codes über die einzelnen Generatoren erfolgte automatisiert über ein Skript-File, so dass bei einer Anpassung einer Konfiguration alle Module sicher neu erzeugt werden können (siehe Abbildung 37).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

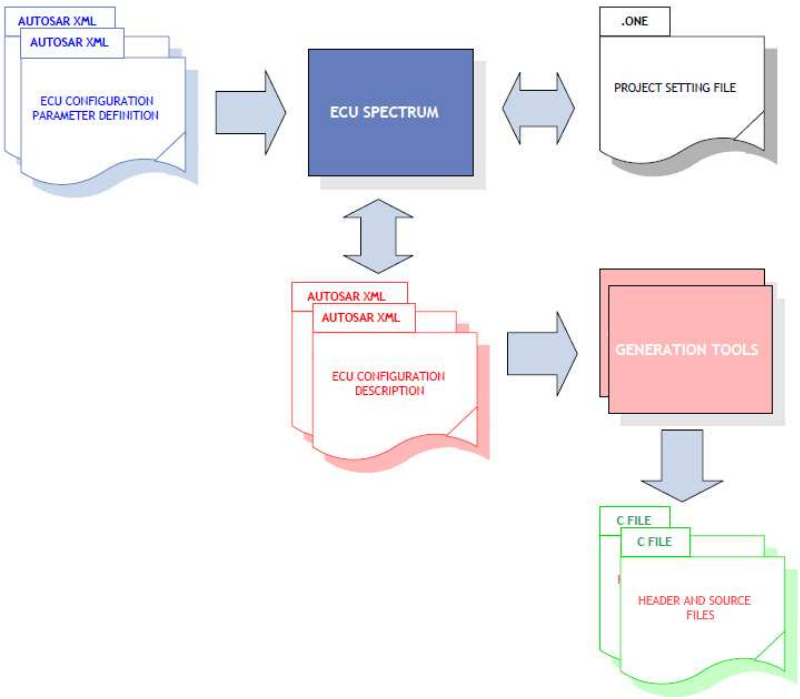


Abbildung 37: Ablauf zum Generieren der AUTOSAR-Treiber [28]

Über das Konfigurations-Tool „ECU-Spectrum“ (siehe Abbildung 38), welches zusammen mit den Treibern ausgeliefert wird, wurden die Einstellungen der Treiber entsprechend der Anforderungen des Universalgateways vorgenommen.

Der **MCU-Treiber** wurde so konfiguriert, dass die interne PLL-Einheit den Quarztakt verdoppelt und der Prozessorkern mit 32 MHz betrieben wird.
Die Konfiguration des **Port-Treibers** und des **DIO-Treibers** erfolgte anhand der Pin-Belegung des Prozessors, so dass diese über symbolische Bezeichner angesprochen werden können (siehe Anhang B).
In dieser Phase der Entwicklung musste auch der **Watchdog-Treiber** konfiguriert werden. Der Watchdog wurde als Standard-Zustand deaktiviert, so dass es bis zur Implementierung des Watchdog-Interfaces und Watchdog-Managers keinen Reset auf Grund eines Überlaufs des Watchdog-Timers geben konnte.
Alle weiteren Treiber wurden nur soweit vorkonfiguriert, dass diese ohne Fehlermeldung in das Projekt eingebunden werden konnten. Im Laufe des Projektes wurde die Konfiguration, bei der Erstellung der entsprechenden Module, an die Erfordernisse angepasst.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

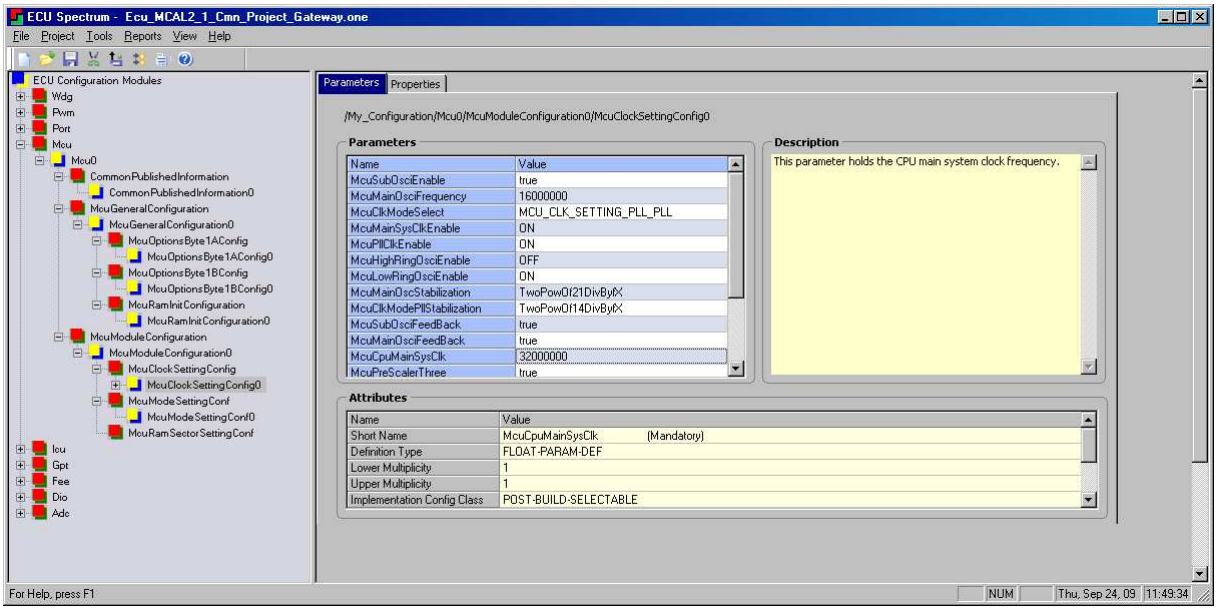


Abbildung 38: Konfiguration MCU-Treiber

Die so erzeugten Treiber wurden im Unterprojekt „MC_Abstraction_Layer“ zusammengefasst und in das Projekt eingebunden. So stehen diese Treiber als Basis für alle weiteren Module der Basis-Software zentral zur Verfügung.

5.5 Systemservices

In den Systemservices sind alle Dienste zusammengefasst, welche für alle Module der Basic-Software notwendig sind. Nachfolgend ist die Implementierung der einzelnen Module dargestellt.

5.5.1 ECU-Manager

Als erstes Modul der Basis-Software wurde der ECU-Manager realisiert. In diesem Modul werden die Zustände des Universalgateways und des Prozessors sowie das Powermanagement verwaltet und gesteuert.

Ein Steuergerät kann entsprechend der AUTOSAR-Richtlinie die Zustände, wie in Abbildung 39 dargestellt, einnehmen. Das Powermanagement des Universalgateways kann jedoch nur das gesamte Steuergerät der Versorgungsspannung trennen (siehe 4.1.1). Durch den Übergang des Prozessors in den Sleep-Mode bei weiterhin aktiver Peripherie werden mit der bestehenden Beschaltung die Anforderungen an das Ruhestromverhalten nicht erfüllt. Im Projekt des Universalgateways wurden daher die Zustände „Wakeup“ und „Sleep“ nicht umgesetzt (grau markiert). Werden diese Zustände im späteren Verlauf des Projektes dennoch

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

benötigt, muss das Powermanagement so angepasst werden, dass die Peripherie und der Prozessor unabhängig voneinander von der Betriebsspannung getrennt werden können.

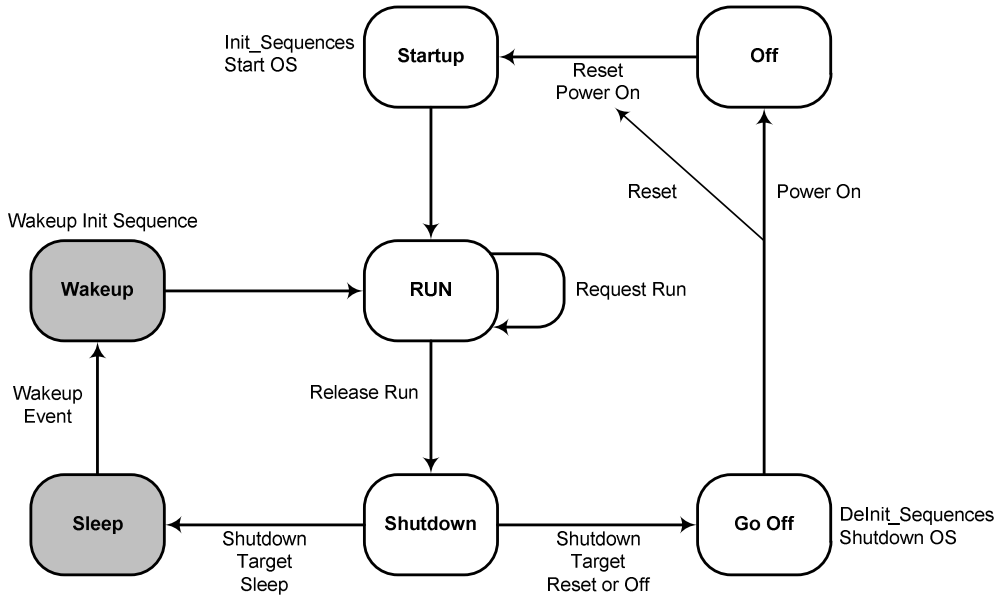


Abbildung 39: Betriebszustände eines Steuergerätes (vereinfacht) [3]

Der in Abbildung 39 dargestellte Zustandsautomat wird innerhalb des Task für den ECU-Manager zyklisch abgearbeitet. Der ECU-Manager prüft dabei selbstständig, ob die Bedingungen für einen Zustandsübergang erfüllt sind. Über einen Funktionsaufruf der RTE ist es auch direkt möglich, den Zustand des ECU-Managers zu beeinflussen (z.B. für einen gezielten Übergang in den Zustand „OFF“). Hat der ECU-Manager einen Zustandswechsel vollzogen, wird dieses über eine Callback-Funktion an die RTE gemeldet.
Der Task des ECU-Managers wird automatisch nach dem Start des Betriebssystems gestartet. In diesem Fall befindet sich der ECU-Manager im Zustand „Startup“. In diesem Zustand wird der RAM des Mikrocontrollers und alle unmittelbar benötigten Module der Basis-Software initialisiert (z.B. der Port-Driver). Ist die Initialisierung abgeschlossen, werden alle zyklischen Tasks (z.B. Applikation, IO_Manager, RTE) des Universalgateways gestartet und der ECU-Manager geht in den Zustand „RUN“ über. Im Zustand „RUN“ prüft der ECU-Manager zyklisch die Bedingungen für das Abschalten des Steuergerätes (z. B. Busruhe auf dem CAN, WBus inaktiv, keine Anforderungen der RTE). Sind alle Bedingungen erfüllt, deinitialisiert der ECU-Manager die Peripherie, fährt das Betriebssystem herunter und schaltet am Ende den Pin für das Powermanagement ab und das System befindet sich nun wieder im Zustand „OFF“.
Für eine bessere Übersicht sind in Abbildung 40 sind die Schnittstellen des ECU-Managers zur RTE und den Modulen des Microcontroller-Abstraction-Layers dargestellt.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

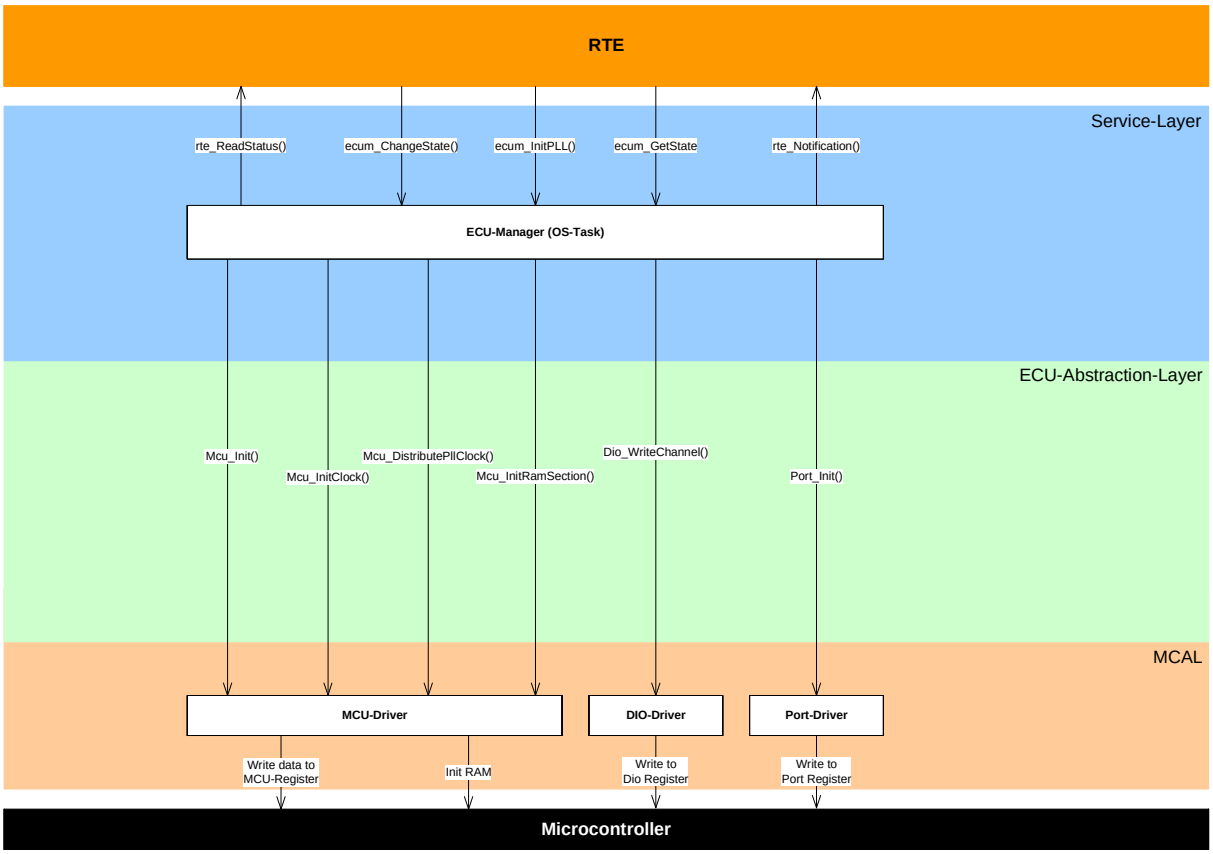


Abbildung 40: Schnittstellen des ECU-Managers

5.5.2 Watchdog

Über den Watchdog soll bei einem Systemfehler eine Neuinitialisierung (Reset) ausgelöst werden. Damit der Watchdog diese Aufgabe bereits während der Initialisierungsphase wahrnehmen kann, wird der Task für den Watchdog unmittelbar nach dem Start des Betriebssystems aufgerufen. Der Start des Watchdog-Task erfolgt auf Grund der hohen Priorität noch vor dem Start des ECU-Managers.

Der Watchdog-Manager initialisiert beim ersten Aufruf des Task das Watchdog-Interface und den Watchdog-Treiber. Der prozessorinterne Watchdog kann über dessen Treiber in drei Modi betrieben werden.

- **Off**: Der Watchdog ist inakitv und muss nicht getriggert werden.
- **Slow-Mode**: Die Zeit bis zum Überlauf des Watchdog-Timers ist sehr lang gewählt. Das Triggern ist nur in sehr groben Schritten notwendig.
- **Fast-Mode**: Die Zeit bis zum Überlauf des Watchdog-Timers ist kurz gewählt. Das Triggern des Watchdogs muss in kurzen Zeitabständen erfolgen.

Im Universalgateway wird der interne Watchdog im Slow-Mode genutzt.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Der Vorteiler für den Timer des Watchdogs im Fast- und Slow-Mode wurden im Konfigurations-Tool des Watchdog-Treibers wie folgt konfiguriert:

- Vorteiler Slow-Mode: $2^{16}/f_{RL}$ (273,1 ms bis Überlauf)
- Vorteiler Fast-Mode: $2^{12}/f_{RL}$ (17,1 ms bis Überlauf)

Dabei wurde als Taktsignal für den Watchdog-Timer das RL-Glied gewählt. Auf diese Weise kann der Watchdog auch bei einem Ausfall des Hauptquarzes weiterarbeiten.

Für den Aufruf des Watchdog-Tasks wurde kein zyklischer Alarm verwendet. Sind die Aufgaben des Watchdog-Managers abgearbeitet, wird der Alarm bis zum nächsten Aufruf des Task mit einer Laufzeit von 10ms gestartet und der Task beendet.

Wird der Task nach Ablauf der 10ms gestartet und Watchdog-Manager aufgerufen, dann werden der interne und der externe Watchdog über das Watchdog-Interface getriggert. Da alle Tasks (bis auf den LCD-Task) nicht präemptiv sind, kann bei einer Fehlfunktion in einem Task der Task des Watchdog nicht mehr gestartet werden. Der Timer des Watchdogs läuft über und löst einen Reset aus. Somit werden keine weiteren Maßnahmen notwendig, damit der Watchdog den Zustand der anderen Task überwachen kann.

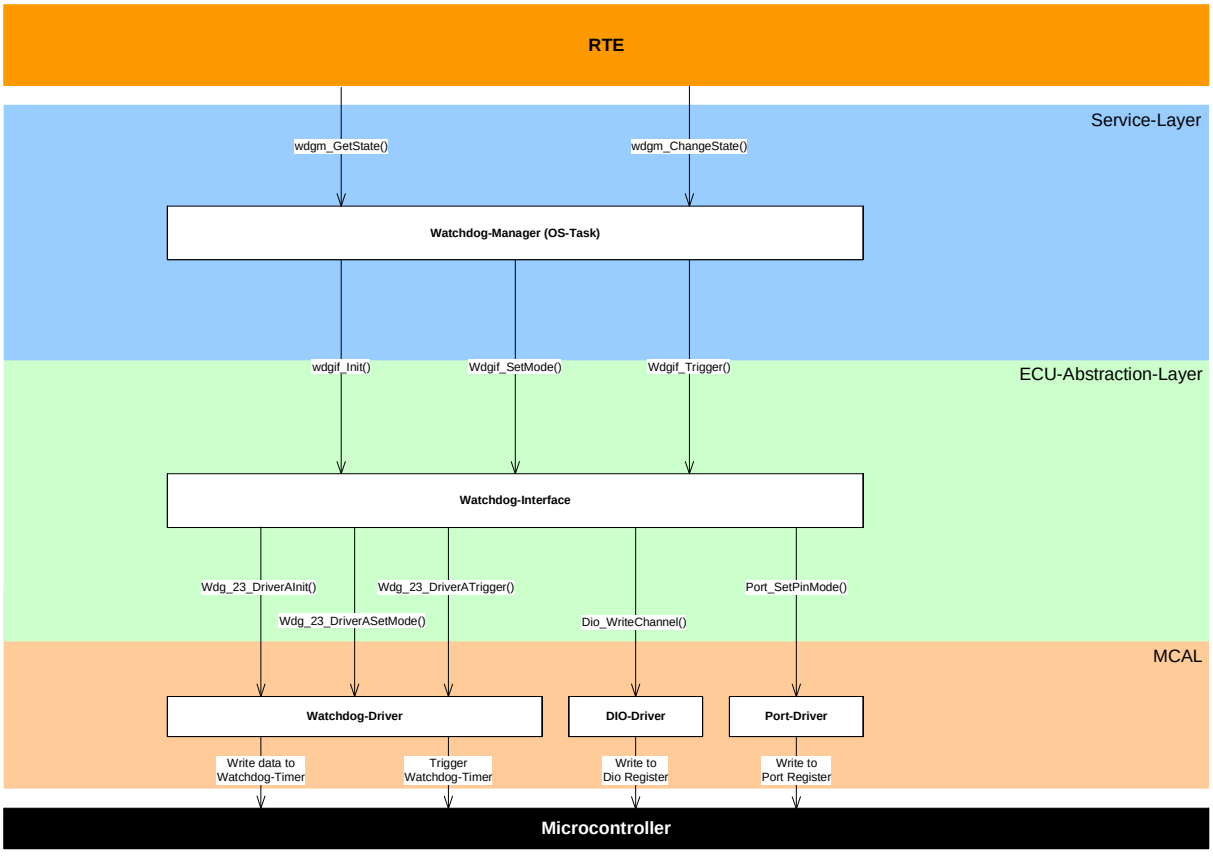


Abbildung 41: Schnittstellen des Watchdogs

Das Watchdog-Interface stellt eine einheitliche Schnittstelle für den internen und externen Watchdog (derzeit nicht in Hardware vorhanden) zur Verfügung. Der Watchdog-Manager ruft lediglich die Funktion „wdgif_Trigger()“ mit dem aktuellen Mode des Watchdogs als Übergabeparameter auf. Das Watchdog-Interface triggert dann entsprechend des internen und externen Watchdog. Die Übergabe des Modes war notwendig, da ein deaktivierter interner Watchdog nicht getriggert werden darf. Der externe Watchdog jedoch muss immer getriggert werden. Es wurde davon ausgegangen, dass extern ein Time-out-Watchdog zum Einsatz kommt, welcher zyklisch über einen Prozessorpin getriggert werden muss (Flankenwechsel). Bei jedem Funtionsaufruf des Watchdog-Interfaces wird daher der Zustand des Prozessor-Pins für den externen Watchdog gewechselt. Über diesen Prozessor-Pin kann mit Hilfe eines Oszilloskops das Zeitverhalten des Watchdog-Task beobachtet und analysiert werden. In Abbildung 41 sind die Schnittstellen des Watchdogs innerhalb der Basis-Software dargestellt.

5.5.3 Timer-Services

Zur Realisierung von Timer für die Basis-Software, welche von Betriebssystem unabhängig sind, wurde das Modul Timer-Services implementiert.
Es können Timer mit einer Zeitbasis von 1 ms und einer maximalen Laufzeit von 49 Tagen (2^{32} ms) realisiert werden. Eine Reduzierung des maximalen Wertes für den Zähler von 32 Bit auf 16 Bit würde die maximale Laufzeit auf 65536 ms (1 Minute, 5 Sekunden) begrenzen. Dies wäre nicht ausreichend.
Ist ein Timer abgelaufen, wird eine entsprechend definierte Callback-Funktion aufgerufen.
Die Basis für die Timer-Services bildet der GPT-Treiber des Microcontroller-Abstraction-Layers, welcher als fertiges Modul vorlag und über das Konfigurationstool zu parametrieren war. Für einen zyklischen Timer wurde der Timer TAA40 des Prozssors im Continuous-Mode mit dem Vorteiler 32 zur Prozessorkern-Frequenz konfiguriert. Bei der Initialisierung der Timer-Services wird der GPT mit einem Vergleichswert 1000 gestartet. Auf diese Weise wird zyklisch alle 1 ms ein Interrupt ausgelöst, welcher durch den GPT-Treiber ausgewertet wird.
Innerhalb der Callback-Funktion (Gpt_TAA40_Notification) des GPT-Treibers wird dann der Timer-Service aufgerufen.
Die Timer selbst werden über ein Array realisiert, welches die einzelnen Zählerstände beinhaltet. Die Bezeichnungen der einzelnen Timer werden als Enumeration im Header-File definiert, so dass global über einen symbolischen Namen auf die entsprechende Position im Array zugegriffen werden kann. Beim Aufruf der Timer-Services durch die Callback-Funktion des GPT wird jede Position des Timer-Arrays dekrementiert. Wurde der Wert Null für einen

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Timer erreicht, löst der Timer-Service die Callback-Funktion aus, welche vorab als Funktions-Pointer festgelegt wurde. Ein Timer mit dem Wert 0 wird nicht weiter bearbeitet.

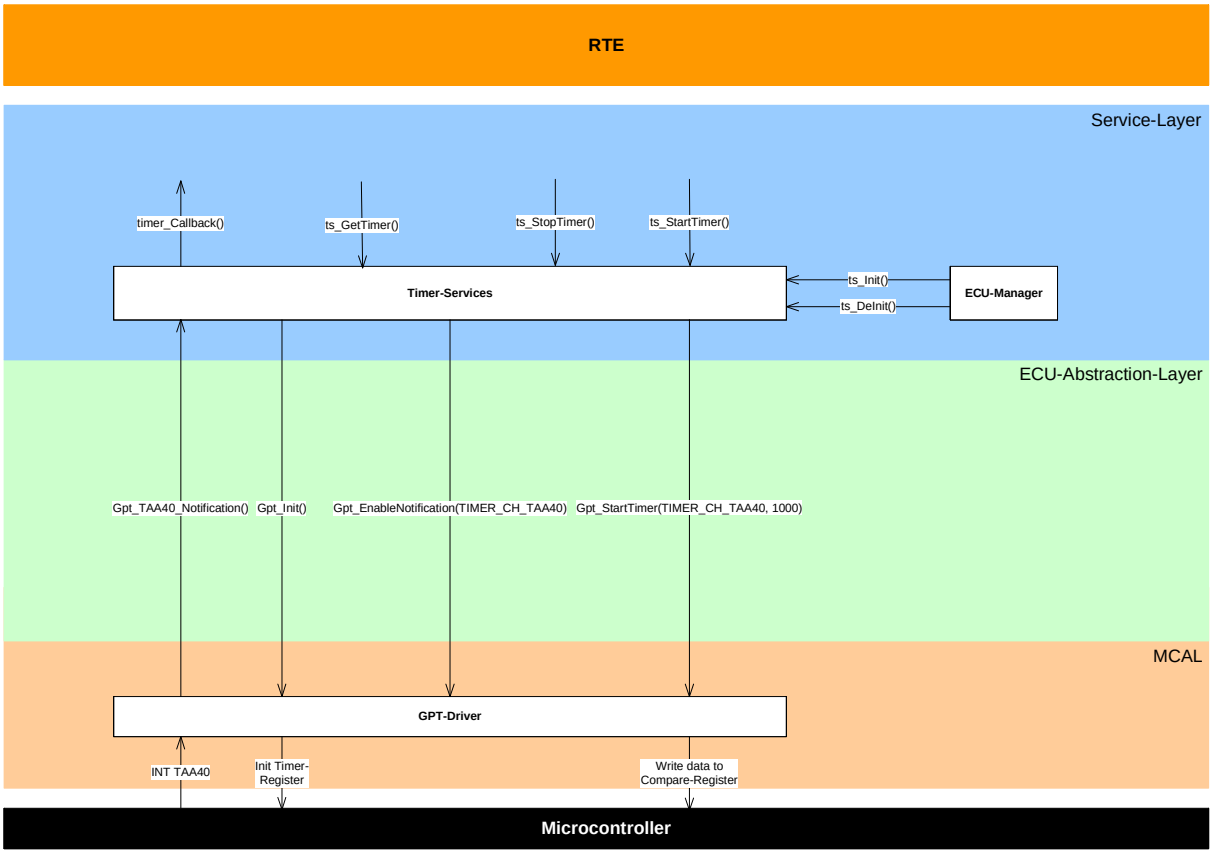


Abbildung 42: Schnittstellen der Timer-Services

Über die Funktion „ts_StartTimer()“ können die einzelnen Timer gestartet werden. Bei Aufruf dieser Funktion wird neben der Bezeichnung des Timers auch die Zeit für den Timer in Millisekunden übergeben. Wird der Wert 0 übergeben oder die Funktion ts_StopTimer() aufgerufen, dann stoppt der entsprechende Timer, ohne die Callback-Funktion aufzurufen. Die aktuelle Zeit (in ms) bis zum Ablauf eines Timers kann über die Funktion ts_GetTimer() abgefragt werden.

In Abbildung 42 sind die Schnittstellen des Timer-Services dargestellt. Innerhalb der Basis-Software nutzt der Treiber für den Wabsto-Bus diesen Timer, Für die Realisierung des Kommunikationsprotokolls (z.B. Interbyte-Timer, Interframe-Timer, Sleep-Timer).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5.6 Input / Output

Das Modul „Input/Output“ der Basis-Software übernimmt die komplette Steuerung und Abstraktion der Ein- und Ausgänge des Universalgateways für die RTE. Dabei greift der IO-Manager auf die Treiber des Microcontroller-Abstraction-Layers zurück, welche als fertige AUTOSAR-Module vorliegen (siehe Abbildung 43).

5.6.1 IO-Manager

Der IO-Manager wird durch den ECU-Manager initialisiert und besitzt einen eigenen zyklischen Task, welcher alle 20 ms vom Betriebssystem aufgerufen wird. Innerhalb dieser Task fragt der IO-Manager die digitalen und analogen Eingänge ab, filtert die Werte und gibt diese zur weiteren Verarbeitung an die RTE weiter.
Benötigt die Applikation Informationen der Ein- oder Ausgänge, so greift diese auf die bereits aufbereiteten Signale, welche vom IO-Manager an die RTE übermittelt wurden, zu.
Nachfolgend ist das Handling des IO-Managements und die Konfiguration der AUTOSAR-Treiber dargestellt.

5.6.2 Port-Treiber

Der Port-Treiber wurde entsprechend der Beschaltung des Mikrocontrollers konfiguriert (siehe Anhang B). Für alle Kommunikationsschnittstellen wurde der Port-Mode als Default-Konfiguration gewählt, da diese Prozessor-Pins zum Teil auch direkt durch Module der Basis-Software angesteuert werden (z.B. bei der Initialisierung des LCD). Die Umschaltung vom Port-Mode auf die erforderliche Kommunikationsschnittstelle erfolgt durch den entsprechenden Kommunikationstreiber über den Funktionsaufruf „Port_SetPinMode()“.
Alle digitalen Ausgänge wurden ohne internen Pullup-Widerstand und mit dem Zustand „Low“ als Default-Wert konfiguriert. Einzige Ausnahme ist der Ausgang für die Steuerung des Powermanagements. Dieser Ausgang wurde mit dem Zustand „High“ als Default-Wert konfiguriert, um nach dem Wecken des Prozessors schnellstmöglich die Selbsthaltung übernehmen zu können (Schaltung siehe Abbildung 25).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5.6.3 Digitale Ein- und Ausgänge

Im DIO-Treiber wurden alle am Prozessor beschalteten Pins mit einem symbolischen Namen versehen (siehe Anhang B), so dass ein einfacher Zugriff des IO-Managers möglich ist.
Innerhalb des IO-Managers werden alle digitalen Eingänge und der Zustand der drei Taster abgefragt. Zur Entprellung der Signale wurde für jeden Eingang ein Counter implementiert. Erst wenn das Signal eines Eingangs 5x hintereinander mit demselben Zustand ausgelesen wird, gilt der Pegelwechsel als bestätigt und die RTE wird über den Zustandswechsel informiert.
Die digitalen Lastausgänge können durch die RTE über die Funktion iom_WriteDigOut() asynchron zum IO-Task geschaltet werden. Die Auswertung der Zustände der Lastausgänge erfolgt ähnlich die Auswertung der digitalen Eingänge. Durch den Vergleich der Ansteuerleitung für den Hardware-Treiber und den Rückmeldeleitungen kann der aktuelle Zustand (Ein, Aus, Unterbrechung, Kurzschluss) der digitalen Ausgänge ermittelt werden (siehe Tabelle 4). Zur Entprellung der Informationen wurde auch hier je Ausgang ein Timer realisiert. Ermittelt der IO-Manager denselben Zustand des digitalen Ausgangs 5x hintereinander, so gilt dieser als bestätigt und wird der RTE mitgeteilt. Auf diese Weise entsteht eine Verzögerung zwischen dem Schalten des digitalen Ausgangs und der Rückmeldung an die RTE von 100 ms.
Wird durch den IO-Manager an einem digitalen Ausgang ein Kurzschluss festgestellt (Ansteuerleitung = High und Rückmeldeleitung = Low), dann wird dieser digitale Ausgang abgeschaltet und die RTE über eine Callback-Funktion informiert.

5.6.4 Analoge Eingänge

Für die Realisierung der analogen Eingänge wurden im ADC-Treiber zwei Kanäle mit folgenden Einstellungen konfiguriert:

- Conversion-Time $512/f_{xp}$ (16 µs)
- Single-Mode / one shot
- Software-Trigger

Das Messen der analogen Werte wird immer am Ende des IO-Task angestoßen. Die Wandlung der Werte erfolgt dann asynchron durch die Hardware des Mikrocontrollers im Hintergrund. Ist die Wandlung eines analogen Eingangswertes abgeschlossen, löst der ADC-Treiber eine Callback-Funktion aus. Innerhalb dieser Callback-Funktion werden die analogen Werte ausgelesen, gefiltert und zwischengespeichert. Beim nächsten Aufruf des IO-Managers werden diese Werte dann an die RTE übermittelt. Auf diese Weise wird das Warten des Tasks auf das Wandlungsergebnis vermieden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Der verwendete Filter basiert auf einem zeitdiskreten Tiefpassfilter aus bestehenden Projekten. [22] Der Filter wurde leicht modifiziert und mit dem Wert 5 für die Filterkonstante implementiert. Wird für die Filterkonstante der Wert 1 gewählt, dann ist der Filter inaktiv.

$$U_{FILTER(n)} = \frac{U_{FILTER(n-1)} * (F-1) + U_{ROHMESS}}{F}$$

U_{FILTER} ist die Ausgangsgröße des Filters, $U_{ROHMESS}$ der Rohmesswert der AD-Wandlung und F ist die Filterkonstante.
Der Zusammenhang zwischen F und der Zeitkonstante τ des Filters lautet:

$$\tau = F * t_{zyklus}$$

Für die Implementierung entspricht t_{zyklus} der Zykluszeit des IO-Task (20 ms), so dass sich für die analogen Eingänge τ =100 ms ergibt.

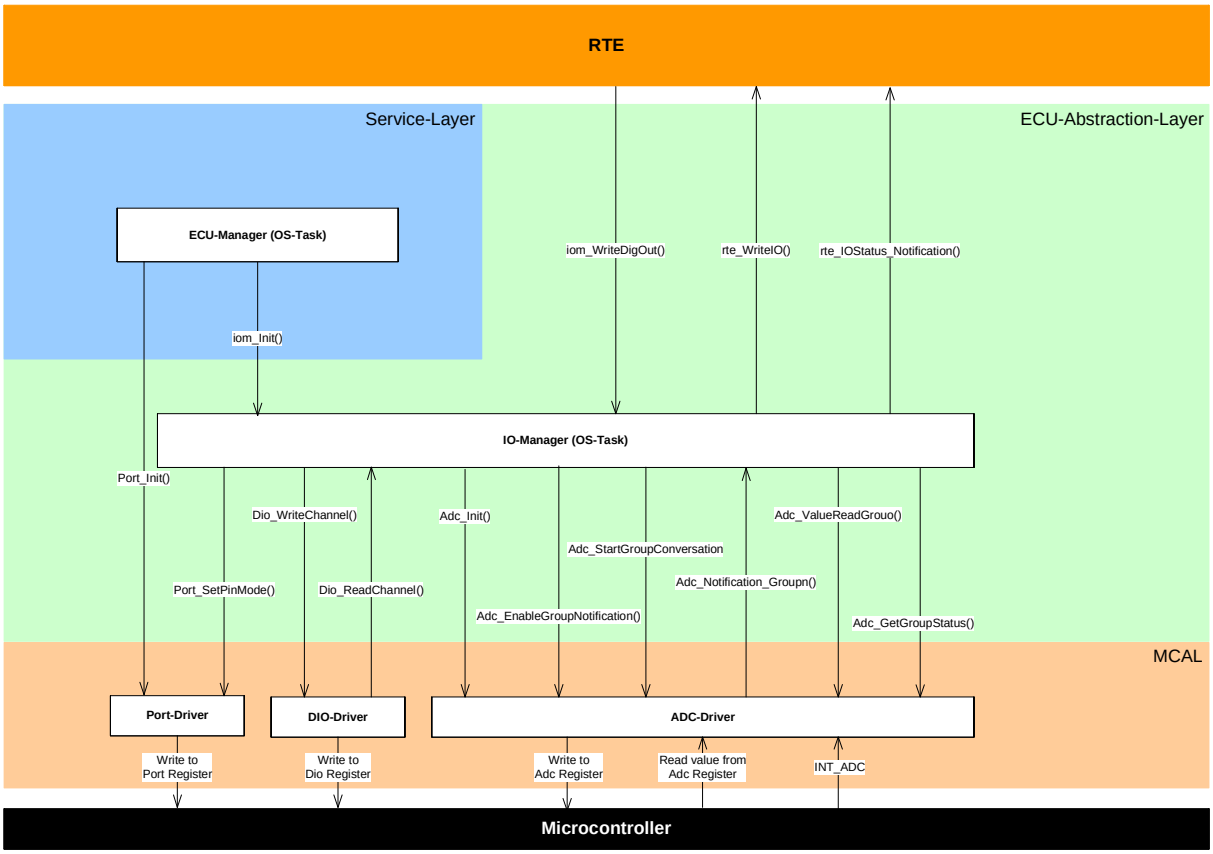


Abbildung 43: Schnittstellen des Input-Output-Managers Blatt 1

Der IO-Manager steuert neben den bisher angesprochenen Ein- und Ausgängen auch noch den PWM-Treiber und den ICU-Treiber.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5.6.5 PWM-Treiber und ICU-Treiber

Über den PWM-Treiber können am Lastausgang des Universalgateways Signale verschiedene Pulsweiten und Frequenzen erzeugt werden. Der PWM-Treiber nutzt dabei vorhandene Hardware-Zähler des Mikrocontrollers.
Der ICU-Treiber kann genutzt werden, um Ereignisse an einem digitalen Eingang zu zählen. Dabei nutzt der ICU-Treiber wie auch der PWM-Treiber die Hardware-Zähler des Mikrocontrollers.
Derzeit besteht keine Anforderung für die Nutzung eines PWM-Signals oder zum Zählen von Ereignissen. Aus diesem Grund wurden lediglich der PWM- und ICU-Treiber implementiert. Die entsprechende Funktionalität des IO-Managers werden implementiert, wenn entsprechende Anforderungen durch die Applikation gestellt werden. Sollte dauerhaft auf die Verwendung dieser Treiber verzichtet werden können, so können diese auf einfache Weise aus dem Projekt entfernt werden.
In Abbildung 44 sind die Schnittstellen des IO-Managers zum PWM- und ICU-Treiber als Übersicht dargestellt.

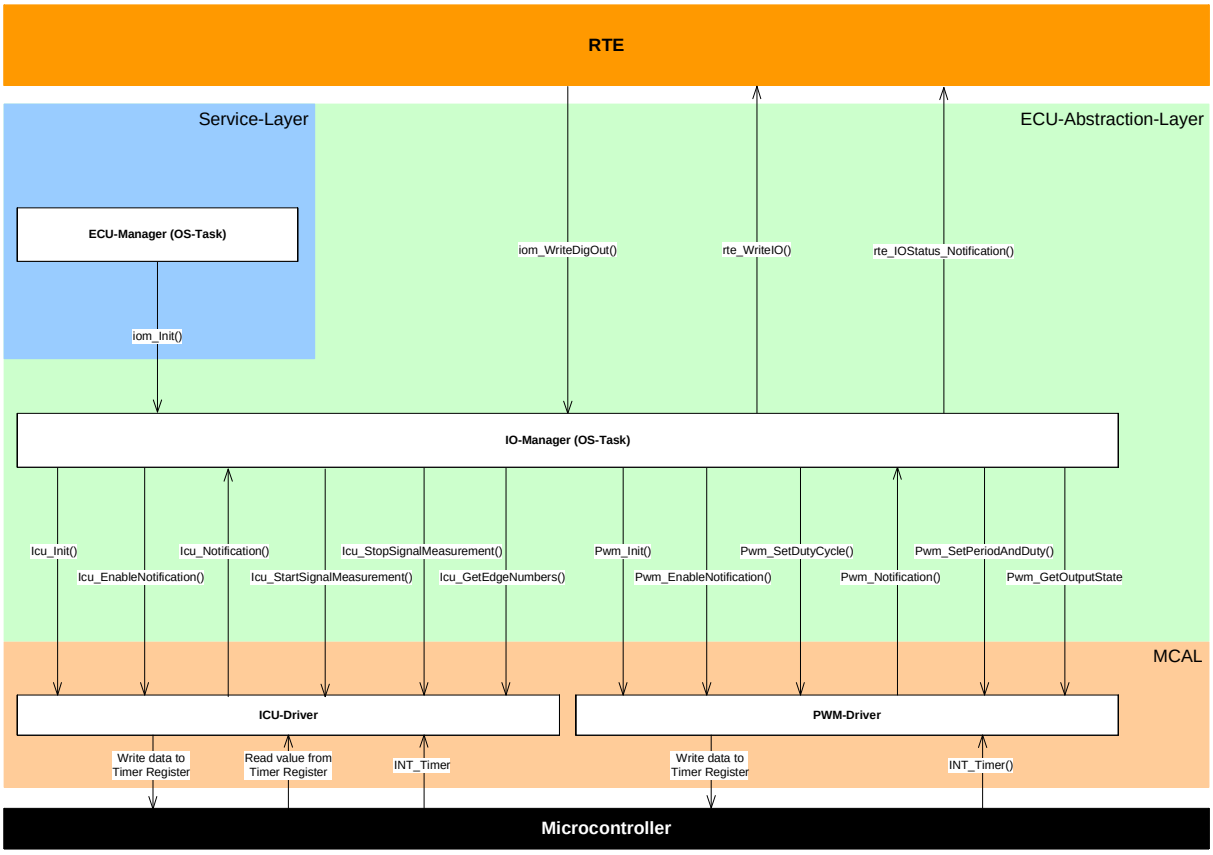


Abbildung 44: Schnittstellen des Input-Output-Managers Blatt 2

5.7 LCD-Anbindung

Als Anzeige wurde für das Universalgateway ein LCD mit 16 Bit Farbtiefe und 176 x 132 Pixel gewählt, welches über eine SPI-Schnittstelle an den Mikrocontroller angeschlossen ist.
Die Anbindung des LCD wurde innerhalb der Complex-Driver in die Software-Architektur integriert (siehe Kapitel 4.2).
Die Steuerung und Datenübermittlung zum LCD erfolgt unidirektional in 16 Bit-Worten in einer festen Konfiguration. Aus diesem Grund wurde auf den Einsatz eines SPI-Treibers, der den vollen Umfang der Funktionalitäten der SPI-Schnittstelle des Mikrocontrollers unterstützt, verzichtet. Der erstellte Treiber initialisiert die SPI-Schnittstellemit folgenden Einstellungen:

- Communication-Typ: 4 (default level “Low” / rising edge)
- Input clock: $f_{xx}/2$ (8 MHz)
- Data lenght: 16 Bit

Über die Funktionen spi_SelectDevice() und spi_DeSelectDevice() wird die Select-Leitung des LCD angesteuert und über die Funktion spi_SendData() werden die Daten synchron zum LCD versandt (d.h. das Programm muss warten, bis der Versandt abgeschlossen ist).
Die Initialisierung des LCD erfolgt in 9 Schritten mit einer Wartezeit von insgesamt 825 ms. Ferner müssen die Zeiten zwischen den einzelnen Initialisierungsschritten mit geringen Toleranzen eingehalten werden. Damit nicht alle anderen Tasks auf die Initialisierung warten müssen und dennoch die erforderlichen Zeiten eingehalten werden, wurde für die Initialisierung des LCD ein eigener Task mit hoher Priorität vorgesehen. Ist ein Initialisierungsschritt abgeschlossen wird der Alarm für die nächste Aktivierung der Task mit der vorgeschriebenen Wartezeit gestartet und der Task beendet. In der Zeit bis zum nächsten Aufruf des Init-Task können andere Tasks abgearbeitet werden.
Ist die Initialisierung des LCD abgeschlossen, wird automatisch der Task für die LCD-Applikation gestartet.
Für die Abschaltsequenz des LCD wurde dasselbe Vorgehen wie für den Start gewählt. Diese Sequenz verläuft in 11 Schritten und benötigt insgesamt 415 ms Laufzeit. Beim Übergang des Systems in den Low-Powermode beendet der ECU-Manager den Task der LCD-Applikation und startet die den LCD-Init-Task, welcher die Sequenz ausführt.
Die einzelnen Sequenzen für die Steuerung des LCD sind in Anhang C aufgeführt.

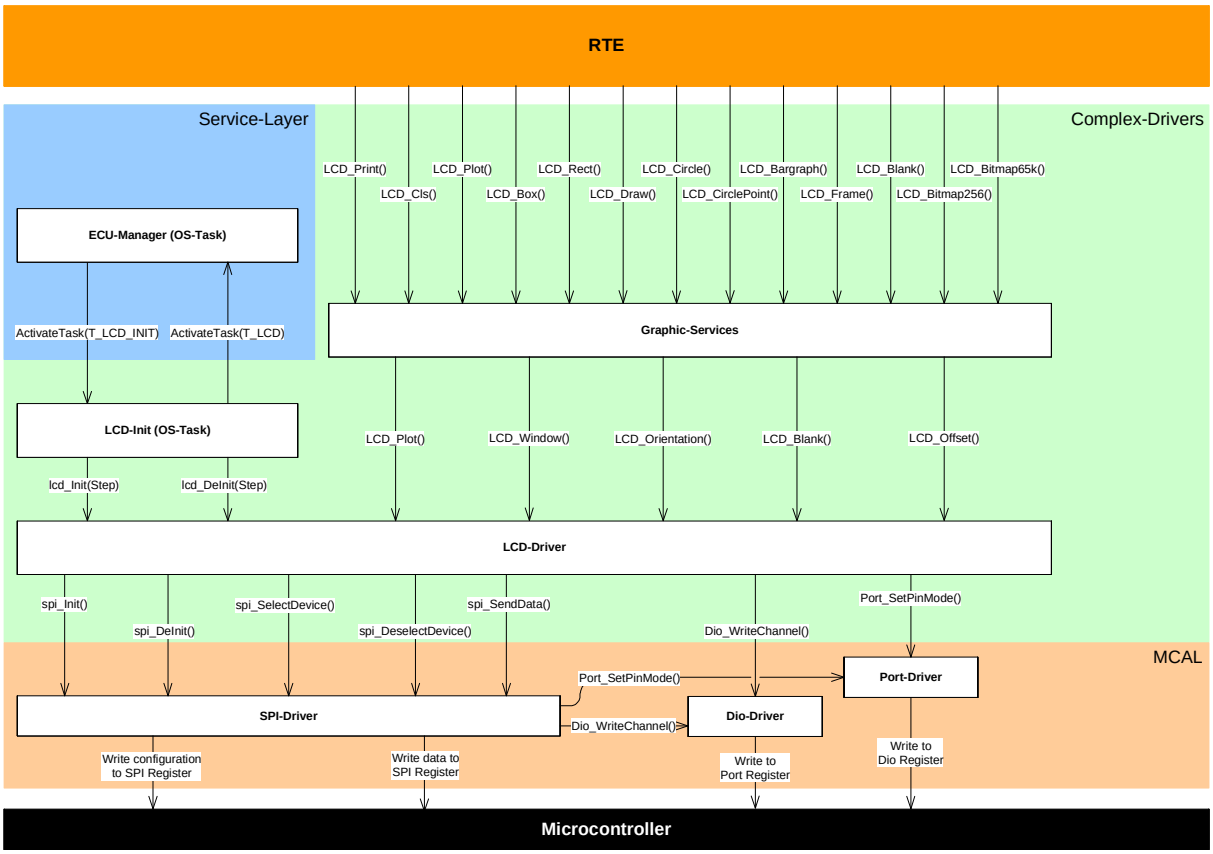


Abbildung 45: Schnittstellen des LCD-Treibers

In der Abbildung 45 sind die Software-Schnittstellen der LCD-Steuerung dargestellt.
Für einige Schritte der Initialisierung des LCD müssen die Pins der SPI-Schnittstelle direkt angesteuert werden. Für die Erfüllung dieser Anforderungen kann die SPI-Schnittstelle über die Funktion „spi_DisableInterface()“ vorübergehend deaktiviert werden.
Der Treiber für die Ansteuerung des LCD ist oberhalb des SPI-Treibers durch den LCD-Driver als Low-Level-Treiber und den Graphic-Services als High-Level-Treiber realisiert.
Der **LCD-Treiber** realisiert neben der Sequenz für die Initialisierung und zum Abschalten des LCD alle Basis-Funktionen, welche für die Darstellung von Informationen notwendig sind.
Diese Funktionen basieren auf dem mit dem Display ausgelieferten Programmierhandbuch [25]. Die dort dargestellten Beispiel-Routinen wurden ursprünglich für den einen Mikrocontroller der Firma Atmel erstellt und mussten für die Nutzung auf dem Fx3-Prozessor angepasst werden.
Mit Hilfe der folgenden 5 Funktionen werden alle Informationen auf dem Display dargestellt.

- Anzeigebereich festlegen (LCD_Window())
- Punkt zeichnen (LCD_Plot())
- LCD-Darstellung verbergen (LCD_Blank())

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

- LCD-Darstellung verschieben (LCD_Offset())
- Darstellungsrichtung auf dem LCD ändern (LCD_Orientation())

Der Ablauf der Datenübermittlung ist dabei immer gleich (siehe Abbildung 46):

1. Anzahl der Pixel für die darzustellenden Daten ermitteln.
2. Setzen der Koordinaten für die Ausgabe mit „LCD_Window()“.
3. Senden der 16 Bit-Informationen für jeden Bildpunkt zeilenweise von links oben nach rechts unten. Wird das Zeilenende erreicht, dann erfolgt automatisch ein Zeilenumbruch (siehe Abbildung 47).

Das Setzen und Rücksetzen der Steuerleitung LCD_DC (Data / Communication ID) und der Select-Leitung für das LCD erfolgt dabei automatisch innerhalb der Funktionen.

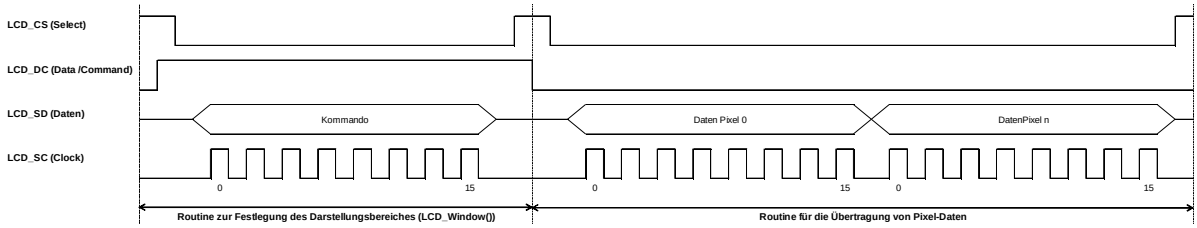


Abbildung 46: Symbolische Darstellung der Datenübertragung zum LCD

Soll beispielsweise das Display in einer bestimmten Farbe gelöscht werden, so muss das gesamte Display ausgewählt werden (176 * 132 Punkte) und im Anschluss muss die Farbinformation über eine Schleife 23232-mal übertragen werden.

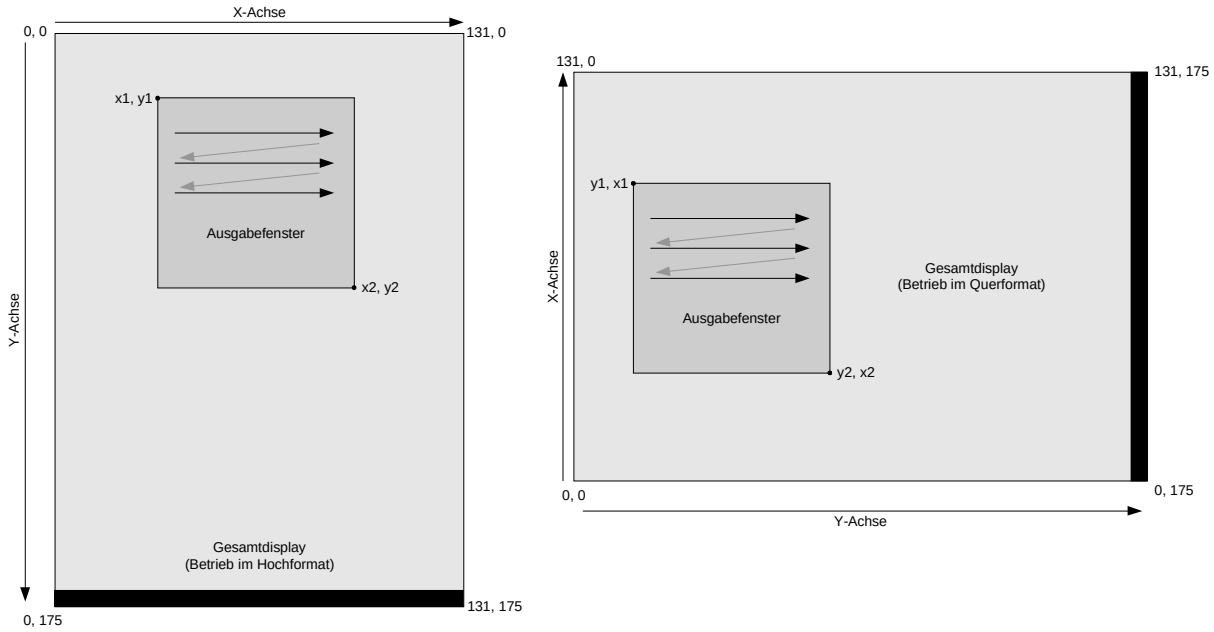


Abbildung 47: Darstellung von Informationen auf dem LCD [25]

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Die Funktionen für die Darstellung komplexer Elemente (z.B. Text, Linie, Kreis) wurden in den **Graphic-Services** zusammengefasst und stehen somit als einfache Funktionsaufrufe für die LCD-Applikation zur Verfügung. Die Graphic-Services selbst sind sehr eng mit den elementaren Funktionen des LCD-Treibers verbunden. Aus diesem Grund wurde der LCD-Treiber und die Graphic-Services in einer Datei (LCD.c und LCD.h) zusammengefasst.
Für die Darstellung einer Linie (waagerecht oder senkrecht) wird ein entsprechend kleines Fenster ausgewählt und mit Pixeln gefüllt. Für einen Kreis oder eine diagonale Linie wird für jeden Punkt ein einzelnes Fenster geöffnet und die Farbinformation übertragen. Die Position für jeden Punkt wird dabei durch einen entsprechende Algorithmus berechnet.
Soll Text oder ein Bild auf dem Display dargestellt werden, so muss nach dem Übermitteln des Anzeigefensters die Farbinformation für jeden Punkt einzeln ermittelt und übertragen werden. Dieser Vorgang benötigt sehr viel Rechenzeit. Aus diesem Grund wurde der Task für die LCD-Applikation auch als präemptiv mit geringer Priorität ausgelegt. So kann die Darstellung auf dem LCD als Hintergrundprozess ablaufen und alle anderen Tasks werden nicht gestört.
Für die Darstellung von Text auf dem Display wurden zwei vorgefertigte Schriften (5 x 8 und 8 x 14 Punkte je Zeichen) integriert [25]. Über die Funktion „LCD_Print()“ können diese zur Darstellung von Text auf dem Display genutzt werden.
Für die Erstellung neuer Schriften können mit Hilfe von Freeware-Tools wie „Font.exe“ auf einfache Weise erstellt werden.
Für die Darstellung von Grafikelementen muss die Bildinformation in einem Array abgelegt werden. Beim Darstellen der Daten auf dem Display wird dann die Information Punkt für Punkt aus diesem Array zum Display übertragen. Da die Darstellung einer Grafik, welche das gesamte Display ausfüllt, 46464 Bytes benötigt, wurde ein Algorithmus implementiert, bei welchem die Möglichkeit besteht, bei der Darstellung eines Bildes aus 256 von 65536 Farben auszuwählen (indizierte Farben wie bei einem gif-Bild). Der Speicherbedarf sinkt dann auf 33026 Bytes.
Die erforderlichen Algorithmen wurden aus dem Programmierhandbuch des Displays übernommen und für das Projekt angepasst [25]. Für die Umwandlung von Grafik-Daten in ein Daten-Array, welches von den Graphic-Services verarbeitet werden kann, wurde zusammen mit dem Display das Programm „Image Converter“ ausgeliefert (siehe Abbildung 48).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

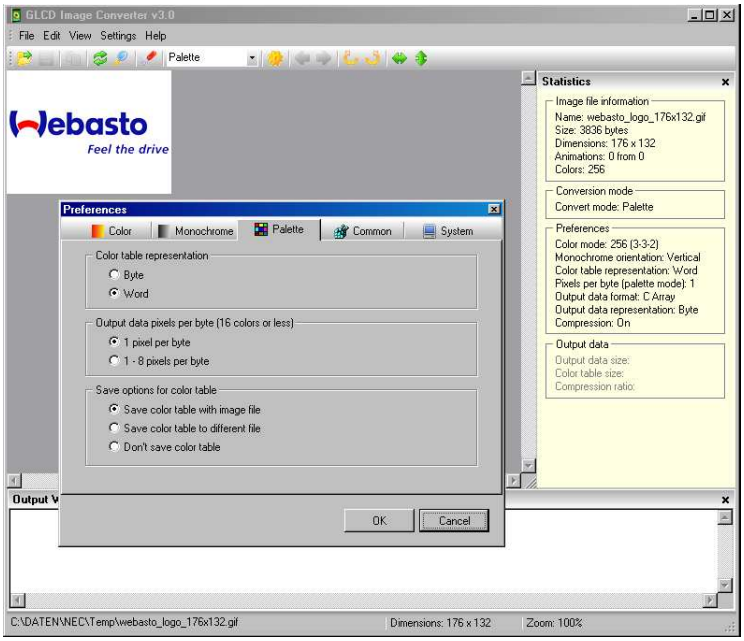


Abbildung 48: Programm zum Konvertieren von Bildern in ein Daten-Array

Mit Hilfe dieses Programms wurden einige Grafik-Elemente (z.B. Kontrollleuchten, Firmenlogo) konvertiert über ein Header-File zusammengefasst und in das Projekt eingebunden. Die Grafik-Elemente wurden mit indizierten Farben (256 aus 65536 Farben) konvertiert. Dabei wurde immer auf die selbe Farbtabelle zurückgegriffen, um weiteren Speicherplatz zu sparen. Für die Darstellung von Statusinformationen wurden Kontrollleuchten in der Größe 23 x 23 und Symbole für die Anzeige des Status der Diagnoseverbindung in der Größe 25 x 25 Punkte konvertiert. Neben den Pixel-Grafiken zur Darstellung von Informationen wurden auf Basis der vom LCD-Treiber zur Verfügung gestellten Funktionen Elemente für die Ausgabe von Daten in Form von Bargraphen oder zum Zeichnen von Umrahmungen erstellt.

Abbildung 49: Grafik-Elemente für die Darstellung von Informationen auf dem LCD

Wo immer möglich, sollte auf die großflächige Nutzung von Pixel-Grafiken verzichtet werden. Die Nutzung von Routinen zum Berechnen und Zeichnen von Elementen ist platzsparender als die Nutzung von Pixel-Grafiken.
Durch die Nutzung der Graphic-Services steht für die LCD-Applikation eine umfangreiche Funktionsbibliothek zur Verfügung, welche bei Bedarf jederzeit erweitert oder angepasst werden kann. Alle Elemente können innerhalb der LCD-Applikation einzeln platziert und dargestellt werden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5.8 Communication Services

Innerhalb der Communication-Services wurden alle Module zur Kommunikation des Universalgateways über Datenbusse zusammengefasst. Nachfolgend ist die Realisierung der Kommunikation über den Webasto-eigenen Bus (WBus) und dem CAN dargestellt.

5.8.1 Webasto-Bus

Der Webasto-Bus (WBus) wurde zur direkten Kommunikation der Webasto-Standheizung mit den Bedienelementen und zur Diagnosekommunikation entwickelt. Er ist als Multi-Master-Bus mit maximal 8 Teilnehmern definiert. Die physikalische Schicht des WBusses entspricht dabei dem LIN (Ein-Draht-Bussystem mit Fahrzeugmasse als Bezugspunkt).
Die Datenübertragung nutzt das UART-Protokoll als Basis, so dass der WBus auf einfache Weise auf fast allen Mikroprozessoren appliziert werden kann.
Die Standard-Geschwindigkeit auf dem WBus beträgt 2400 bit/s (8 Datenbits, even Parity und einem Stoppbit), so dass der Protokoll-Stack oberhalb der UART-Schnittstelle in Software implementiert werden kann.

Der Aufbau eines Frames auf dem WBus ist in Abbildung 50 dargestellt. Die Länge eines Frames beträgt mindestens 4 Byte und kann maximal 100 Byte betragen. Empfangene Frames werden innerhalb der WBus-Services dekodiert und bei einer Sendeaufforderung wird dort der komplette Datenrahmen zusammengestellt. Alle anderen Module des WBus-Routers haben auf den Inhalt und Aufbau eines Frames keinen Einfluss.

Byte 0	Byte 1	Byte 2	Byte 3			
Sender/ Empfänger	Länge des Frames	Service-ID	Daten-Byte 0	…	Daten-Byte n	Check-summe

Abbildung 50: Aufbau eines Datenrahmens auf dem WBus

Alle bisherigen bei Webasto eingesetzten Treiber für den WBus sind auf die Kommunikation über einen einzelnen Bus ausgelegt. Für die Kommunikation mit mehreren Teilnehmern auf verschiedenen Bussen oder für das Routing von Frames zwischen den Bussen wäre die Anpassung der bestehenden Treiber sehr aufwendig gewesen. Aus diesem Grund wurde für das Universalgateway ein eigener Treiber erstellt. Für eine bessere Übersichtlichkeit und Strukturierung wurde der Code für den WBus entsprechend seiner Aufgaben in einzelne Module aufgeteilt (siehe Abbildung 51). Der Aufbau der dargestellten Module stellt gleichzeitig die Hierarchie der Module innerhalb des Systems dar.

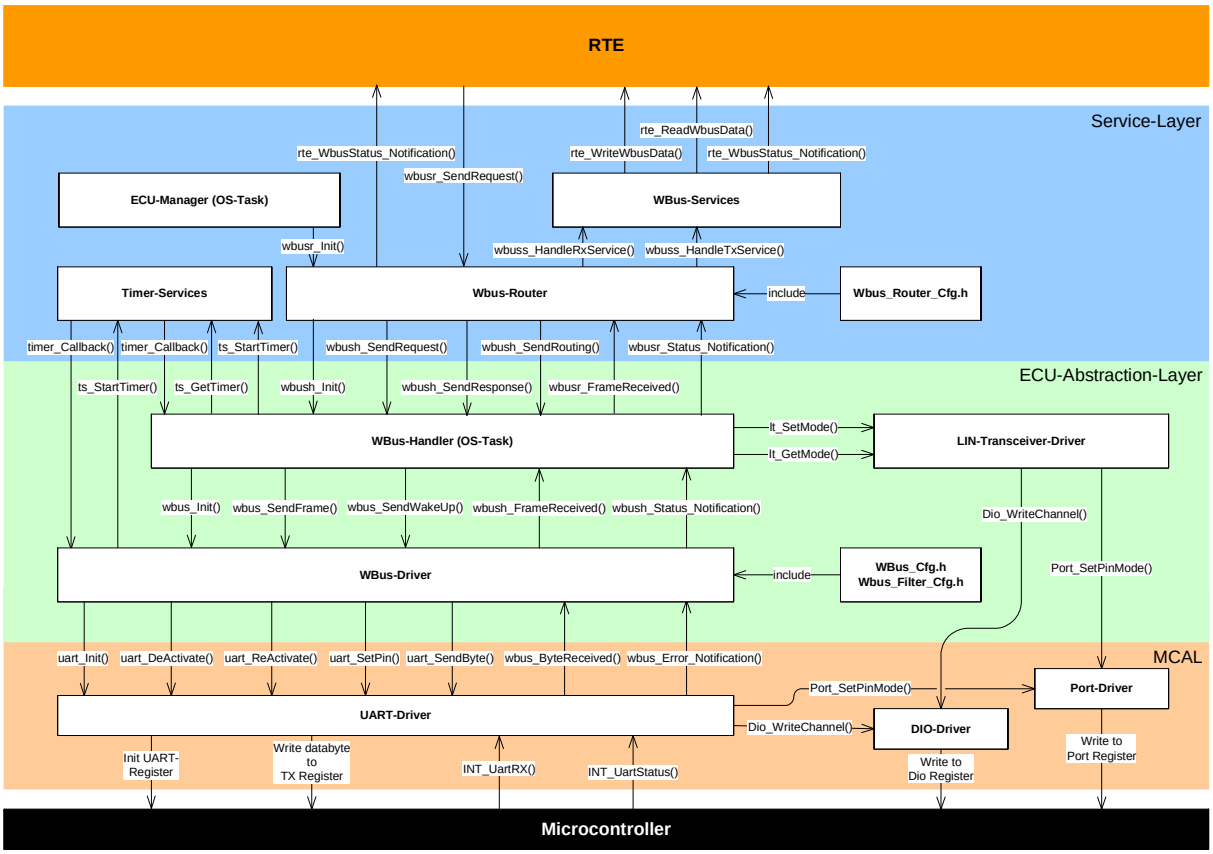


Abbildung 51: Interfaces des WBus-Routers

Als erstes Modul für den WBus wurde der **UART-Treiber** erstellt. Der Treiber für die serielle Schnittstelle übernimmt die Aufgabe, Einzelbytes ,welche vom WBus-Treiber übergeben werden, auf den Bus zu senden und Einzelbytes vom Bus zu empfangen, diese auf Fehler (Paritätscheck) zu prüfen und an den WBus-Treiber weiterzureichen.

Basisadresse: 0xFFFFFA00
Offset zur nächsten Schnittstelle: 0x20

UART Register	Offset UART Register
UDnCTL0	0x00
UDnCTL1	0x01
UDnCTL2	0x02
UDnOPT0	0x03
UDnSTR	0x04
UDnOPT1	0x05
UDnRX	0x06
UDnTX	0x07

Tabelle 7: Basis-Adresse und Offset UART-Register

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Für das Offset der einzelnen Register der Schnittstelle und den Zugriff auf die einzelnen Konfigurations-Bits wurden Strukturen als Datentyp definiert.
Beim Aufruf einer Funktion zum Zugriff auf die serielle Schnittstelle wird immer die UART-Nummer mit übergeben. Basierend auf dem bekannten Aufbau der Register wird dann die Adresse für den Zugriff auf die Konfigurationsregister nach folgendem Schema berechnet:

Adresspointer = Basisadresse + (UART-Nummer * Offset zur nächsten Schnittstelle)

Die ermittelte Adresse wird einem lokalen Pointer (uart_module_ptr) übergeben, welcher als Typ der Registerstruktur definiert wurde. Auf diese Weise kann recht einfach auf die benötigten Register und Konfigurationsbits aller 8 Schnittstellen zugegriffen werden.
Nachfolgend ist als Beispiel der Zugriff auf die Konfigurations-Register innerhalb der Initialisierungs-Phase dargestellt.

```
uart_module_ptr -> ctl0.pwr     = OFF;          /* UART deaktivieren */
uart_module_ptr -> ctl1         = speed;        /* base clock setzen */
uart_module_ptr -> ctl2         = BAUDRATECLOCK;/* baud rate clock setzen*/
uart_module_ptr -> ctl0.dir     = direction;    /* Datenrichtung setzen */
uart_module_ptr -> ctl0.ps      = parity;       /* Parity setzen */
uart_module_ptr -> ctl0.cl      = lenght;       /* Datenlänge setzen */
uart_module_ptr -> ctl0.sl      = stop;         /* Stopp-Bits setzen */
uart_module_ptr -> ctl0.rxe     = ON;           /* TX-Funktion aktivieren */
uart_module_ptr -> ctl0.txe     = ON;           /* RX-Funktion aktivieren */
uart_module_ptr -> str.overRunError = OFF;      /* Overrun-Fehler löschen */
uart_module_ptr -> str.framingError = OFF;      /* Framing-Fehler löschen */
uart_module_ptr -> str.parityError  = OFF;      /* Parity-Fehler löschen */
uart_module_ptr -> str.noiseFlag    = OFF;      /* Noise-Fehler löschen */
uart_module_ptr -> ctl0.pwr = ON;               /* UART aktivieren */
```

Neben dem Zugriff auf die Register der seriellen Schnittstelle muss der UART-Treiber auch die direkte Steuerung des Tx-Pins ermöglichen. Diese Funktion wird für die Wakeup-Sequenz des Busses benötigt. Über die Funktion „uart_DeActivate()“ kann dazu die Schnittstelle vorübergehend deaktiviert und die Prozessor-Pins in den Port-Mode versetzt werden. Die Funktion „uart_SetPin()“ ermöglicht den direkten Zugriff auf die Tx-Leitung. Die Reaktivierung der Schnittstelle erfolgt über die Funktion „uart_ReActivate()“.

Der **WBus-Treiber** baut auf dem UART-Treiber auf und ist Bestandteil des ECU-Abstraction-Layers. Der WBus-Treiber übernimmt komplette Daten-Rahmen vom WBus-Handler und übergibt diese in Einzelbytes an die serielle Schnittstelle. Das Senden eines Frames erfolgt asynchron zum weiteren Programmablauf. Es wird lediglich das erste Byte des Frames direkt an den UART-Treiber übergeben. Alle weiteren Bytes werden innerhalb der Interrupt-Routine des UART-Treibers versandt. Systembedingt wird jedes Byte, welches auf dem Bus versandt wird, direkt wieder empfangen. Im WBus-Treiber wird diese Eigenschaft genutzt, um eine Datenkollision auf dem Bus zu erkennen. Stimmt das empfangene Byte nicht dem versandten Byte überein, wird dies als Fehler erkannt, das Senden eingestellt und der WBus-Handler informiert. Nach Auflauf einer festgelegten Wartezeit stößt der WBus-Handler dann die Sendewiederholung an.
Empfängt die serielle Schnittstelle ein Datenbyte, wird dieses an den WBus-Treiber durch die Funktion „wbus_ByteReceived()“ übergeben. Dieser sammelt die Daten innerhalb eines Empfangs-Puffers und startet bei jedem Byte den Interbyte-Timer mit einer Laufzeit von 5 ms neu. Der Interbyte-Timer kann nur komplett ablaufen, wenn kein Byte mehr empfangen wird. Ist dies der Fall, muss sich im Empfangs-Puffer ein kompletter Datenrahmen befinden. Um sicherzustellen, dass dieses Frame gültig ist, wird die Checksumme und die Länge des Frames geprüft (Vergleich von Byte 1 des Frames mit der der Anzahl empfangender Bytes). Ferner wurde ein Filter integriert, damit an die übergeordneten Schichten nur die Frames weitergereicht werden, welche im Universalgateway genutzt werden. Der Filter ist als Array realisiert, bei welchem die Adresse des Empfängers der Position im Array entspricht. Über die Parameter „TRUE“ oder „FALSE“ kann für jeden möglichen Empfänger die Akzeptanz im Header-File „wbus_Filter_Cfg.h“ konfiguriert werden. Wird bei der Abfrage des Arrays an der Position des Empfängers der Wert „TRUE“ erkannt, dann wird dieses Frame über die Funktion „wbush_FrameReceived()“ an den WBus-Handler weitergereicht.
Neben diesen Aufgaben steuert der WBus-Treiber auch die Wakeup-Sequenz für den Bus. Die Wakeup-Sequenz wird vom WBus-Handler angestoßen und läuft über einen Timer gesteuert asynchron zum restlichen Programmablauf. Ist die Sequenz abgeschlossen, erfolgt eine Rückinfo über die Funktion wbush_Status_Notification() an den WBus-Handler, welcher dann das zu sendende Frame an den Treiber übergibt (wbus_SendFrame()).
Für das Handling der einzelnen Busse wurden der Empfangs- und der Sendepuffer als zweidimensionales Array angelegt (Nummer des Busses / Größe des Empfangspuffers) und je Bus ein Satz Timer konfiguriert. Für die korrekte Steuerung beim Senden und Empfangen von Daten wird immer die Nummer des Busses beim Aufruf der entsprechenden Funktion mit übergeben. Auf diese Weise können mehrere Busse parallel gehandhabt werden.

Zur einfachen Konfiguration des WBus-Treiber wurden die Parameter zur Steuerung (z.B. Timing, Bezeichnung der Busse, Anzahl der verwendeten Busse) in das Header-File „WBus_Cfg.h“ ausgelagert.

Der **WBus-Handler** steuert den Ablauf der Kommunikation auf dem WBus. Für diese Ablaufsteuerung wurde innerhalb der WBus-Task ein entsprechender Zustandsautomat realisiert (siehe Anhang D). Der Task des WBus-Handlers wird nicht zu festen Zykluszeiten aktiviert. Die Aktivierung erfolgt immer dann, wenn über einen Funktionsaufruf eines der folgenden Ereignise eintritt:

- Es liegt eine Sendeaufforderung vor.
- Die Wakeup-Sequenz des Busses ist abgeschlossen.
- Es wurde ein Frame erfolgreich versandt.
- Das Frame wurde beim Senden zerstört.
- Es wurde ein Frame empfangen.
- Der Timer für Sendewiederholung ist abgelaufen.
- Der Interframe-Timer ist abgelaufen.
- Der Sleep-Timer ist abgelaufen.

Tritt eines der dargestellten Ereignisse ein, wird das entsprechende Event-Flag und das Bearbeitungs-Flag des entsprechenden Busses gesetzt und der Task des WBus-Handlers aktiviert. Wenn der Task des WBus-Handlers vom Betriebssystem in den Zustand „RUN“ versetzt wird, prüft eine Routine die Bearbeitungs-Flags aller Busse. Ist ein Bearbeitungs-Flag gesetzt, dann wird dieses Flag zurückgesetzt und der Handler des entsprechenden Busses gestartet. Dieser prüft anhand der Event-Flags, ob ein Zustandswechsel erforderlich ist und leitet bei Bedarf die empfangenen Frames an den WBus-Router weiter (wbusr_FrameReceived()). Bei einer Sendeaufforderung des WBus-Routers stößt der Handler das Wecken des Busses an und prüft den Versandt des Frames (evtl. ist der Bus besetzt). Konnte das Frame nicht versandt werden (z.B. bei einer Datenkollision auf dem Bus) oder wird zum versandten Frame keine Antwort empfangen, dann erfolgt automatisch eine Sendewiederholung bis die maximale Anzahl von Wiederholungen erreicht ist. Wurde ein Frame erfolgreich versandt oder ist beim Senden ein Fehler aufgetreten, dann wird dieses dem WBus-Router über eine Callback-Funktion mitgeteilt.
Auch der WBus-Handler wurde mehrdimensional gestaltet, damit jeder Bus einzeln bearbeitet werden kann.

Die Verwendung des empfangenen Frames wird innerhalb des **WBus-Routers** anhand des adressierten Empfängers ermittelt. In einer Routing-Tabelle kann dazu für jeden Empfänger der Busse ein Ziel-Bus (WBus 0 … WBus n), die interne Verwendung oder das Verwerfen des Frames definiert werden. Der Aufbau der Routing-Tabelle ähnelt dem Filter-Array des WBus-Treibers. An der Position im Array, welche der Adresse des Empfängers entspricht, ist der Zielbus des Frames angegeben (WBus 0 … WBus n). Wenn ein gültiger Zielbus konfiguriert wurde, dann wird das Frame über die Funktion „wbush_SendRouting()“ an den entsprechenden WBus-Handler übergeben. Ist an der entsprechenden Position der Bezeichner „INTERN“ konfiguriert, wird das Frame über die Funktion wbuss_HandleRxService() an die WBus-Services weitergereicht. Ferner besteht die Möglichkeit, ein Frame, welches auf einen anderen Bus geroutet wird, via Monitorfunktion intern auszuwerten (Bit 7 an der Position des Empfängers = 1).
Erfolgt über die Funktion „wbusr_SendRequest()“ durch die RTE eine Sendeaufforderung an den WBus-Router, so wird das entsprechende Frame innerhalb der WBus-Services zusammengestellt. Der zu sendende Service wird dabei zurammen mit der Angabe des Zielbusses als Parameter mit übergeben. Ist das Frame zusammengestellt, wird dieses durch den WBus-Router an den WBus-Handler übergeben (wbush_SendRequest()). Kann aktuell kein Frame auf dem Bus versandt werden, weil der Bus nicht bereit ist, dann wird dies als negative Antwort an die RTE gemeldet. Ist der Bus wieder frei, so erfolgt automatisch der Versandt des Frames.

Innerhalb der **WBus-Services** wird ein empfangenes Frame dekodiert oder bei einer Sendeaufforderung das entsprechende Frame zusammengestellt. Handelt es sich um eine Anfrage nach Daten, dann wird automatisch das erforderliche Antwort-Frame zusammengestellt. Beim Funktionsaufruf „wbuss_HandleRxService()“ wird dazu ein Zeiger auf den entsprechenden Sendepuffer des WBus-Routers übergeben. Die für das Antwortframe erforderlichen Daten werden über die RTE bezogen. Ist das Frame zusammengestellt und die Checksumme berechnet, wird dieses Frame durch den WBus-Router über die Funktion „wbush_SendResponse()“ versandt. Handelt es sich bei dem empfangenen Frame um ein Daten-Frame, dann wird dieses dekodiert und die Daten an die RTE übermittelt.
Bei einer Sendeaufforderung werden über die Funktion „wbuss_HandleTxService()“ der geforderte Service und der Zeiger auf den Sendepuffer übergeben. Durch die WBus-Services wird das angeforderte Frame zusammengestellt, die Checksumme berechnet und in den Sendepuffer kopiert. Der Versandt dieses Frames wird analog zum Versand eines Antwort-Frames über den WBus-Router und -Handler gesteuert.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Zur korrekten Steuerung des **LIN-Transceivers** wurde ein Treiber erstellt, welcher den Zustandsautomaten entsprechend der Vorgabe aus dem Datenblatt realisiert (siehe Abbildung 52). Zum Umschalten der Zustände des LIN-Transceivers wird neben den Steuerleitungen „NWAKE“ und „NSLP“ auch der TXD-Pin der UART-Schnittstelle benötigt. Aus diesem Grund wird der Treiber für den LIN-Transceiver durch den WBus-Handler angesteuert. Dieser stellt sicher, dass in der Zeit des Umschaltens kein Frame auf den Bus versandt werden kann.
Als Standard-Betriebszustand wird der LIN-Transceiver innerhalb der Initialisierungs-Phase des WBus-Routers in den „Low-Slope-Mode“ versetzt (flachere Flanken für ein verbessertes EMV-Verhalten). Ein Umschalten innerhalb des laufenden Betriebes ist nicht erforderlich. Der Treiber wurde so gestaltet, dass dieser selbst den Prozessor-Pin der seriellen Schnittstelle in den Port-Mode umschalten kann. Auf diese Weise muss die serielle Schnittstelle und der WBus-Treiber nicht deaktiviert werden.

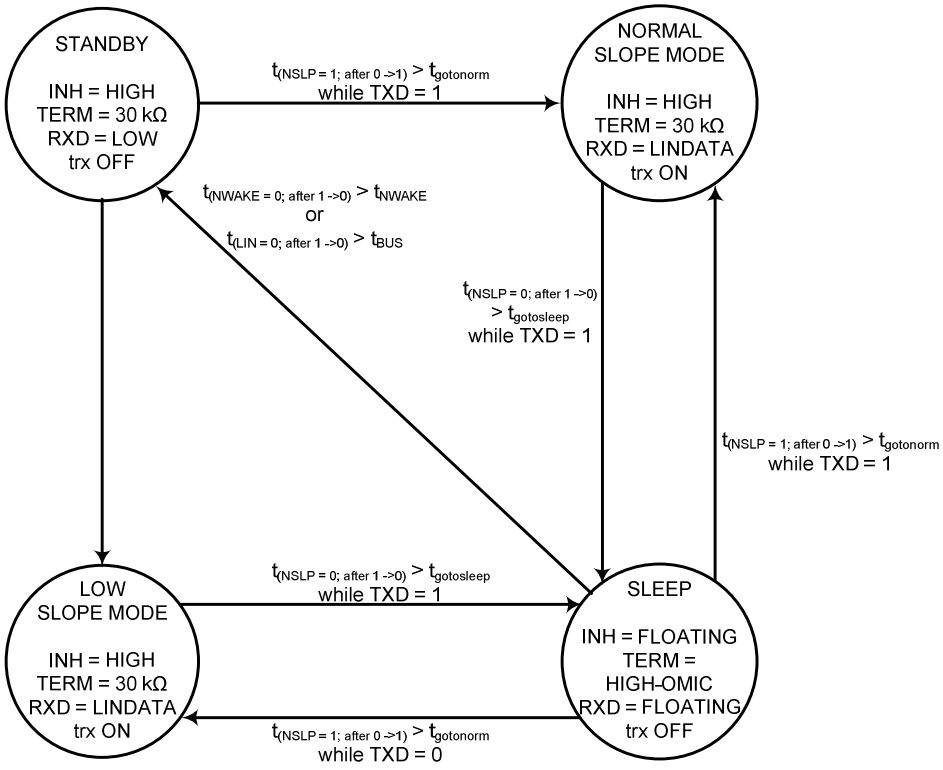


Abbildung 52: Zustände des LIN-Transceivers [24]

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5.8.2 Routen von Daten auf dem WBus

Durch die beschriebene Realisierung kann der WBus-Router kurze Frames (< 30 Byte) zwischen den einzelnen Bussen vermitteln (OSI-Schicht 3). Auf dieser Ebene werden die Frames ausschließlich anhand der Adressierung geroutet. Der Inhalt der Frames kann nicht verändert oder ergänzt werden. Ist es erforderlich, dass Daten manipuliert oder ergänzt werden (z.B. wenn zwei verschiedene WBus-Versionen zum Einsatz kommen), dann muss das Routing innerhalb der Applikation oder der RTE erfolgen (OSI-Schicht 7).

Das Routing von Frames zwischen den Bussen ist in der Spezifikation des WBus so nicht vorgesehen. Die vorgeschriebene Zeit zwischen dem Versand einer Anfrage und dem Empfang der entsprechenden Antwort ist in der Spezifikation mit 500 ms festgelegt. Bei einer Busgeschwindigkeit von 2400 bit/s wird für den Versand eines Bytes ca. 4,6 ms benötigt. Wird ein Frame von einem Bus auf einen anderen geroutet, so erhöht sich die Laufzeit vom Senden des Frames durch die Quelle bis zum Empfang des Frames an der Senke um die Zeit, welche das Frame auf dem Bus benötigt zuzüglich der Mindestzeit zwischen den Frames auf dem Bus (entspricht der Zeit für die Übertragung eines Bytes). Das Frame muss vom Gateway erst komplett empfangen und geprüft werden, bevor es auf dem Ziel-Bus versandt werden kann. Auch die Antwort des Empfängers muss wieder geroutet werden.

- ⇨ Laufzeit ohne Routing = 2x Interbyte-Zeit + Bearbeitungszeit + Framezeit (Antwort)
- ⇨ Laufzeit mit Routing = 4x Interbyte Zeit + Framezeit (Anfrage) + Bearbeitungszeit + 2x Framezeit (Antwort)

Für ein Anfrage-Frame mit 30 Byte Länge und einem entsprechenden Antwort-Frame ergibt sich ohne Beachtung der Bearbeitungszeit im Empfänger und ohne Routing eine Laufzeit von mindestens 146 ms. Beim Routing ergibt dies eine Laufzeit von mindestens 430 ms. Bezieht man die Bearbeitungszeit durch Empfänger und Router mit ein, bedeutet dies, dass die maximale Antwortzeit von 500 ms überschritten werden kann.

Abbildung 53 zeigt die zeitliche Verzögerung beim Routen eines 4 Byte langen Frames auf einen anderen Bus (27 ms). In Abbildung 54 erhöht sich die Verzögerung zusätzlich um die Zeit, welche für die Wakeup-Sequenz benötigt wird auf 80 ms. Die gesamte Laufzeit der Datenkommunikation über zwei Busse (Anfrage + Antwort) inklusive der Verarbeitungszeit innerhalb des Empfängers beträgt ohne Wakeup-Sequenz bei diesem 4 Byte langen Frame 83,2 ms (siehe Abbildung 55).

Die Abweichungen zwischen den gemessenen Laufzeiten und den rechnerisch ermittelten Werten ergeben sich aus der nicht mit einbezogenen Bearbeitungszeit innerhalb des Mikrocontrollers und der Zeit, welche zwischen dem Versand der einzelnen Bytes vergeht.

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

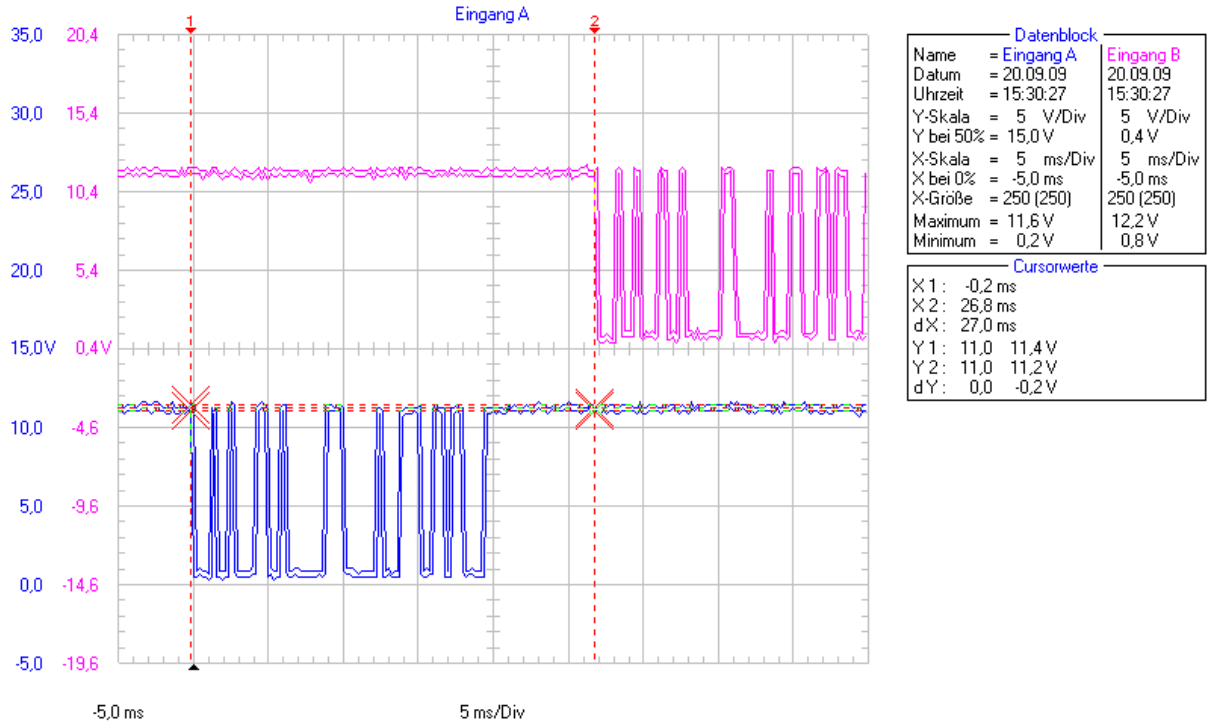


Abbildung 53: Verzögerung beim Routing auf dem WBus

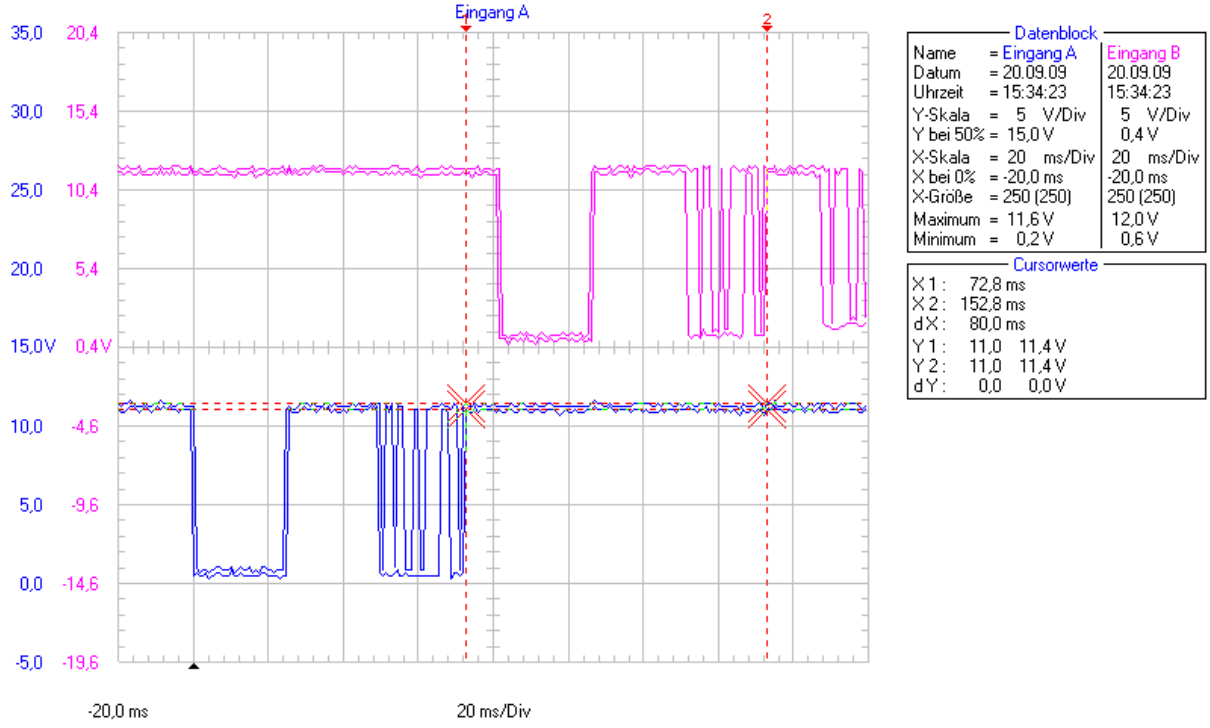


Abbildung 54: Verzögerung beim Routing auf dem WBus mit Wakeup

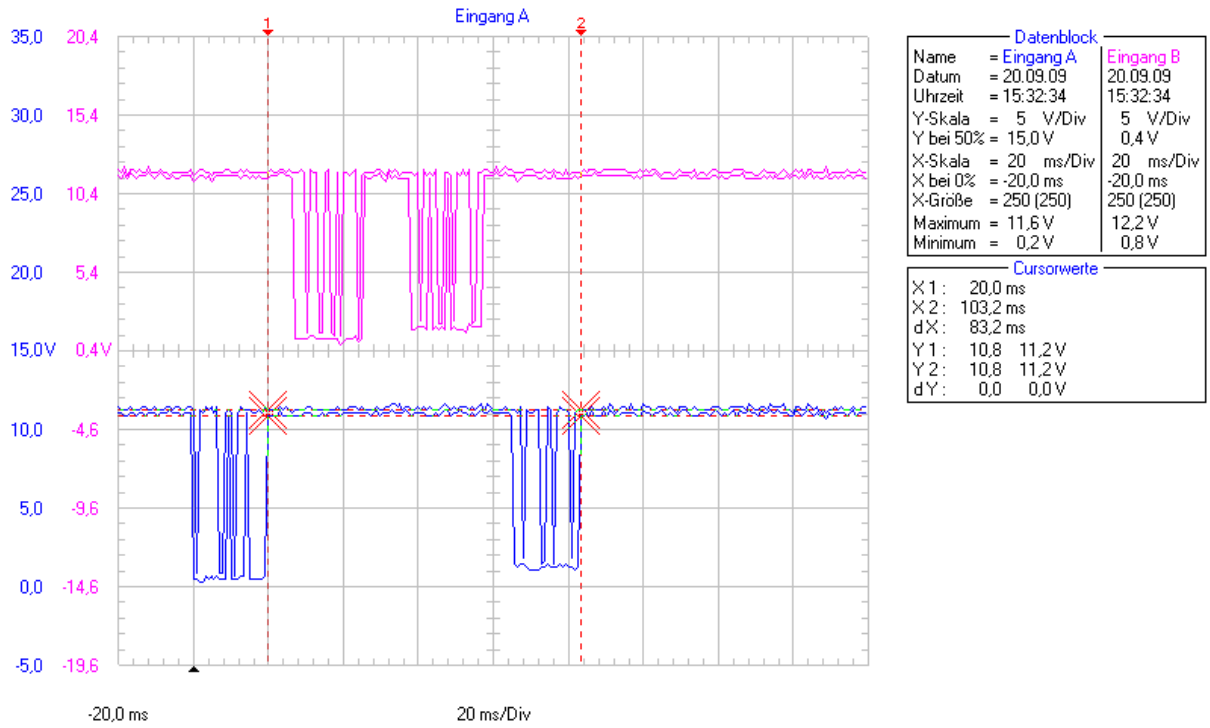


Abbildung 55: Laufzeit zwischen Anfrage und Antwort beim Routing

5.8.3 CAN-Bus

Anders als bei WBus und LIN ist der CAN ein Broadcast-System, bei dem der Sender seine Botschaften mit maximal 8 Byte Nutzdaten ohne Zielangabe absendet. Jede Botschaft besitzt einen Identifier, anhand dessen jeder Teilnehmer die für sich notwendigen Informationen herausfiltert und verarbeitet. Der Mikrocontroller besitzt für die Realisierung des CAN-Protokolls (Data-Link-Layer) eine eigene Hardware. Dieser CAN-Controller realisiert dabei den kompletten Protokoll-Stack in Hardware, wodurch der Mikrocontroller entlastet wird. Die Software muss nicht den Aufbau eines kompletten CAN-Frames beherrschen. Es müssen lediglich die für eine CAN-Botschaft wichtigen Daten gehandhabt werden (Typ des Frames, Identifier, Data-Length-Code (DLC), Daten-Bytes). Der in der Diplomarbeit verwendete V850 FK3 Mikrocontroller besitzt 5 unabhängige CAN-Controller (Aufbau siehe Abbildung 56), welche über die entsprechenden Register konfiguriert werden können. Für das Universalgateway werden davon derzeit 2 CAN-Controller genutzt. Im Aufbau der Software ist eine Erweiterung auf aller 5 CAN-Schnittstellen jederzeit möglich.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

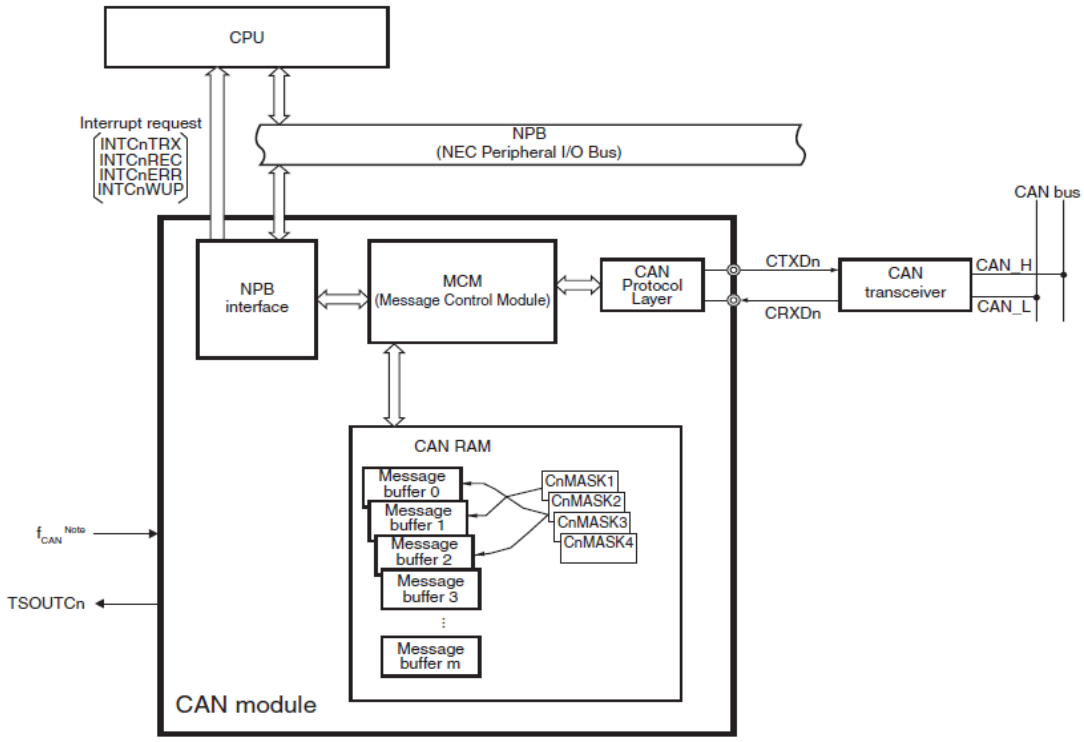


Abbildung 56: Block-Diagramm des CAN-Controllers [26]

Die CAN-Controller verfügen über je 32 Botschafts-Puffer. Jeder Puffer kann dabei zum Senden oder Empfangen von CAN-Nachrichten benutzt werden. Zur Realisierung eines Akzeptansfilters beim Empfangen von Botschaften können vier Mask-Register konfiguriert und den Botschaftspuffern zugewiesen werden.

Für die Kommunikation auf dem CAN gibt es zwei Arten von Registern je CAN-Controller. Über die Konfigurationsregister werden die Eigenschaften des CAN parametriert (z.B. Übertragungsgeschwindigkeit, Abtastzeitpunkt) und die Kommunikation gesteuert (z.B. Statusanzeige, Interrupt-Steuerung). Über die Botschafts-Puffer können Botschaften auf den CAN versandt und empfangen werden.
Für das Universalgateway stand ein Demo-Treiber zur Verfügung, welcher die wichtigsten Funktionen zum Versenden und Empfangen von Botschaften bereits realisierte. Dieser Treiber wurde auf die 5 CAN-Schnittstellen erweitert und in den Funktionen so angepasst und erweitert, dass ein einfacher Zugriff auf die Eigenschaften und Daten möglich wurde. Ähnlich wie bei den UART-Schnittstellen besitzen die Register der CAN-Controller zueinander ein festes Offset, ausgehend von einer Basisadresse. Somit konnte die Berechnung der Speicheradresse für den Zugriff auf die CAN-Register analog zum Vorgehen beim UART-Treiber erfolgen (siehe Kapitel 5.8.1).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Adresspointer = Basisadresse + (CAN-Nummer * Offset zur nächsten Schnittstelle)

Basisadresse: 0x03FEC000

CAN Register	**Offset**
CAN0 registers	0x0000
CAN0 message buffers	0x0100
CAN1 registers	0x0600
CAN1 message buffers	0x0700
CAN2 registers	0x0C00
CAN2 message buffers	0x0D00
CAN3 registers	0x1200
CAN3 message buffers	0x1300
CAN4 registers	0x1800
CAN4 message buffers	0x1900

Tabelle 8: Basis-Adresse und Offset CAN-Register

Die einzelnen Register eines CAN-Controller und dessen Konfigurations-Bits wurden als Struktur definiert. Beim Aufruf einer Funktion zum Zugriff auf den CAN wird immer die Nummer des entsprechenden CAN-Controllers mit übergeben und daraus die Adresse der Register berechnet. Diese Adresse wird einem Pointer zugewiesen, welcher als Struktur der Register definiert wurde. Auf diese Weise ist ein einfacher Zugriff auf die CAN-Register, deren Botschafts-Daten und Konfigurations-Bits möglich.

Der Treiber wurde in zwei Schichten aufgeteilt. Innerhalb der unteren Treiberschicht des CAN-Treibers wurde der Zugriff auf die Register abstrahiert und ein einfacher Zugriff auf die Konfigurations-Parameter realisiert. Zu dieser Treiberschicht gehören beispielsweise folgende Funktionen:

- can_SetCanSpeed()
- can_GlobalOperate()
- can_EnableInt() / can_DisableInt()
- can_EnableTransmission() /can_DisableTransmission()
- can_Transmit()
- can_IsTransmissionCompleted()
- can_SetMask()

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Innerhalb der oberen Treiberschicht des CAN-Treibers wurden diese Elementarfunktionen zu komplexeren Funktionen zusammengefasst, auf welche durch übergeordnete Schichten des CAN-Moduls zugegriffen wird. Zu dieser Schicht gehören:

- can_InitCANChannel()
- can_SendMessage()
- canh_MessageReceived()
- canh_Error_Notification()

Die Parameter zur Initialisierung der Konfigurationsregister, der Filtermasken, die Festlegung der Message-Puffer als Sende- oder Empfangspuffer sowie der Identifier erfolgt über das Header-File CAN_Cfg.h. Eine Anpassung der Konfiguration im laufenden Betrieb (z.B. zum Umschalten der Übertragungsgeschwindigkeit) kann über eine Neuinitialisierung des CAN-Controllers erfolgen. Eine Anpassung der Filtermasken im laufenden Betrieb ist derzeit nicht vorgesehen.

Die Parameter für die in der Automobilindustrie üblichen Übertragungsgeschwindigkeiten und des Abtastzeitpunktes wurden ermittelt und als symbolische Bezeichner im Header-File des CAN-Treibers definiert (siehe Tabelle 9). Nachfolgend ist kurz die Vorgehensweise zur Ermittlung dieser Parameter dargestellt.

Basis für die Übertragungsgeschwindigkeit auf dem CAN ist das Taktsignal, mit welchem der CAN-Controller betrieben wird. Über die Register BRP (bit-rate prescaler register) und BTR (bit-rate register) können die Einstellungen parametriert werden. Anders als bei der UART-Schnittstelle wird so nicht nur die Übertragungsgeschwindigkeit sondern auch der Abtastzeitpunkt des CAN-Signals bestimmt. Der Abtastzeitpunkt soll sich dabei stets im hinteren Drittel der Bitzeit befinden [3]. Es gilt für die Berechnung:

$$Zeitquanten/Bit(TQ) = \frac{f_{Controller}}{CAN-Speed * Vorteiler}$$

$$Zeitquanten/Bit = Sync + TSEG1 + TSEG2$$

$$Abtastzeitpunkt = Sync + TSEG1$$

Für eine Übertragungsgeschwindigkeit von 500 kbit/s, einer Taktfrequenz des Controllers von 16 MHz und einem Vorteiler mit dem Wert 2 (BRP-Register) ergeben sich 16 Zeitquanten je Bit und ein Abtastzeitpunkt bei 81,3% mit TSEG1 = 11 TQ und TSEG2 = 4 TQ (siehe Abbildung 57).

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Die Werte für Sync (Synchronisation), TSEG1 (Zeit-Segment 1) und TSEG2 (Zeit-Segment 2) werden im BTR-Register zusammengefasst. [26].

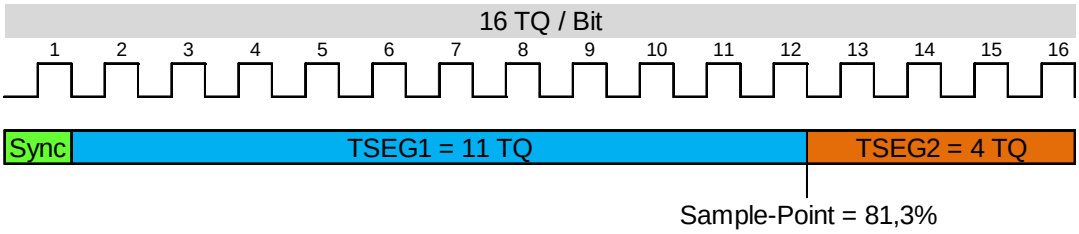


Abbildung 57: CAN-Bit-Timing

Für das Universalgateways wurde ein Abtastzeitpunkt von 81,3% gewählt. Für diesen Abtastzeitpunkt kann der Wert für das BTR-Register für alle üblichen Übertragungsgeschwindigeiten konstant gehalten werden. Es muss für die Wahl der Geschwindigkeit lediglich der Vorteiler über das BTR-Register so angepasst werden, dass sich immer 16 Zeitquanten je CAN-Bit ergeben. Bei einer Taktfrequenz von 16 MHz für den CAN-Controller ergeben sich folgende Werte für das BRP und das BTR-Register:

Übertragungsgeschwindigkeit	**BRP Register**	**BTR-Register**
33.3 kbit/s	0x1D (f_{in}/48)	0x020B
83.3 kbit/s	0x0B (f_{in}/12)	0x020B
100 kbit/s	0x09 (f_{in}/10)	0x020B
125 kbit/s	0x07 (f_{in}/8)	0x020B
250 kbit/s	0x03 (f_{in}/4)	0x020B
500 kbit/s	0x01 (f_{in}/2)	0x020B
1 Mbit/s	0x00 (f_{in}/1)	0x020B

Tabelle 9: Timing-Parameter für die CAN-Kommunikation

Jeder der fünf CAN-Controller besitzt vier Register, mit welchem eine Vorfilterung der Botschafts-Identifier empfangener Botschaften möglich ist. Diese Mask-Register können so konfiguriert werden, dass genau eine Botschaft, eine Gruppe oder alle Botschaften akzeptiert werden. Für jeden Message-Puffer kann ein Empfangs-Identifier eingestellt werden. Wird eine Botschaft empfangen, dann wird der Empfangs-Identifier mit dem im Puffer eingestellten Identifier verglichen. Stimmt dieser überein, dann wird die Botschaft akzeptiert und der Mikrocontroller über den Empfang informiert. Über den Akzeptanzfilter können einige Stellen des Empfangs-Identifier aus dieser Bewertung ausgeschlossen werden, indem das entsprechende Bit auf den Wert 1 gesetzt wird.

Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Beispiel1: Genau ein Identifier wird akzeptiert. Empfangene Botschaft wird akzeptiert.

Akzeptanzfilter	=	0	0	0	0	0	0	0	0	0	0	0
Identifier des Message-Buffers	=	0	1	1	0	0	0	1	0	1	0	1
empfangene CAN-Botschaft	=	0	1	1	0	0	0	1	0	1	0	1
Ergebnis		1	1	1	1	1	1	1	1	1	1	1

Beispiel2: Genau ein Identifier wird akzeptiert. Empfangene Botschaft wird abgelehnt.

Akzeptanzfilter	=	0	0	0	0	0	0	0	0	0	0	0
Identifier des Message-Buffers	=	0	1	1	0	0	0	1	0	**1**	0	1
empfangene CAN-Botschaft	=	0	1	1	0	0	0	1	0	**0**	0	1
Ergebnis		1	1	1	1	1	1	1	1	0	1	1

Beispiel3: Eine Gruppe von Identifier wird akzeptiert. Empfangene Botschaft wird akzeptiert.

Akzeptanzfilter	=	0	0	0	0	0	0	0	0	1	1	1
Identifier des Message-Buffers	=	0	1	1	0	0	0	1	0	x	x	x
empfangene CAN-Botschaft	=	0	1	1	0	0	0	1	0	**1**	**0**	**1**
Ergebnis		1	1	1	1	1	1	1	1	1	1	1

Beispiel4: Eine Gruppe von Identifier wird akzeptiert. Empfangene Botschaft wird abgelehnt.

Akzeptanzfilter	=	0	0	0	0	0	0	0	0	1	1	1
Identifier des Message-Buffers	=	0	1	1	0	0	0	1	**0**	x	x	x
empfangene CAN-Botschaft	=	0	1	1	0	0	0	1	**1**	**0**	**0**	**1**
Ergebnis		1	1	1	1	1	1	1	0	1	1	1

Durch die geschickte Wahl der Werte für die Akzeptanzfilter und Empfangspuffer werden nur die Botschaften an den Mikrocontroller gemeldet, welche dieser auch benötigt.
Derzeit besteht keine Anforderung, spezielle Botschaften über den CAN zu empfangen. Aus diesem Grund sind die Werte für die Akzeptanzfilter aller CAN-Controller mit 1 konfiguriert (alle Botschaften werden akzeptiert).
Wird eine Filterung benötigt, kann der gewünschte Wert im Herade-Filer „can_Cfg.h“ parametriert werden.

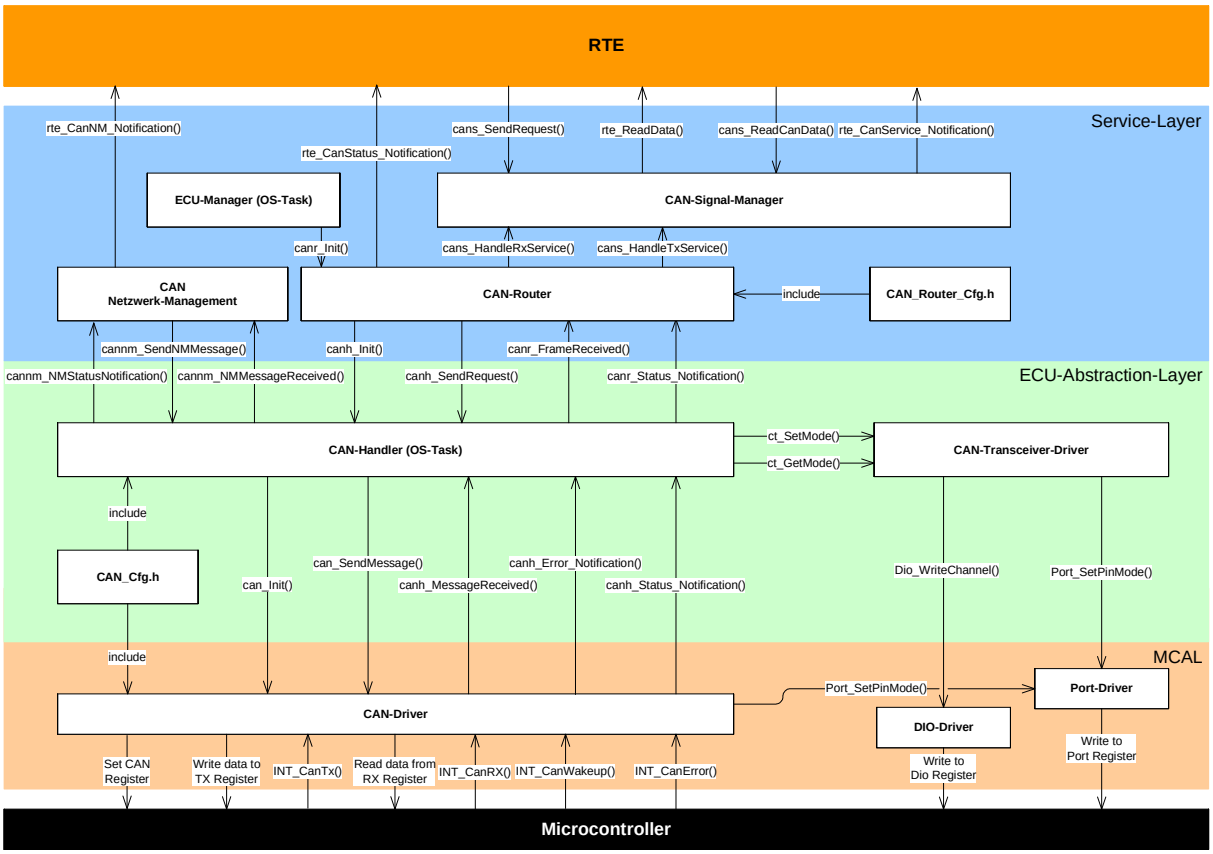


Abbildung 58: Interfaces des CAN-Routers

Auf dem CAN-Treiber baut der **CAN-Handler** auf (siehe Abbildung 58). Der CAN-Handler steuert das Versenden und den Empfang der CAN-Botschaften. Anders als beim WBus-Handler übernimmt der CAN-Controller den Protokoll-Stack. Der Schwerpunkt des CAN-Handlers ist somit die Steuerung des CAN-Treibers und das Fehlerhandling (z.B. bei einer Datenkollisionen auf dem Bus) sowie die Steuerung des Treibers für den CAN-Transceiver (derzeit nicht implementiert). Tritt ein Fehler beim Senden (Error-active) oder Empfangen (Error-passive) einer CAN-Botschaft auf, dann wird im CAN-Controller ein Fehlerzähler inkrementiert (CnERC). Übersteigt der Fehlerzähler für das Senden einen festgelegten Schwellwert, schaltet der CAN-Controller ab und meldet dieses Ereignis an den Mikrocontroller. Das derzeitig umgesetzte Fehlerhandling beschränkt sich darauf, den CAN-Controller neu zu initialisieren und das Ereignis an die übergeordnete Schicht zu melden, wenn der CAN-Controller das Abschalten des CAN veranlasst. Damit der CAN-Handler unabhängig arbeiten kann wurde ein eigener Task realisiert. Ähnlich wie beim WBus-Handler ist dieser Task ereignisgesteuert. Neben dem einfachen Weiterreichen von Botschaften zwischen dem CAN-Router und dem CAN-Treiber werden durch den CAN-Handler die Botschaften für das aktive Netzwerkmanagement herausgefiltert und entsprechend weitergeleitet.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Die Konfiguration, welcher Bereich der Botschafts-Identifier für das Netzwerkmanagement genutzt wird, kann über das Header-File „CAN_Cfg.h“ definiert werden.

Der Treiber für den **CAN-Transceiver** wurde derzeit noch nicht implementiert. Auf der aktuell zur Verfügung stehenden Hardware ist der Transceiver fest angeschlossen, so dass dieser automatisch nach dem Anlegen der Versorgungsspannung in den aktiven Mode übergeht. Die Entwicklung dieses Treibers erfolgt mit der Erstellung der endgültigen Hardware für das Universalgateway.

Für das Management des Netzwerkes (CAN-NM) gibt es verschiedene Möglichkeiten (z.B. OSEK-NM). Der Einsatz des entsprechenden **Netzwerkmanagements** hängt vom Einsatz des Universalgateways ab. Wird das Universalgateway auf einem aktiven Fahrzeugbus eingesetzt, besteht die einfachste Art des Netzwerkmanagementes darin, die Aktivität auf dem Bus zu überwachen (mitlauschen). Wird innerhalb einer bestimmten Zeit keine Botschaft mehr empfangen, dann muss der CAN den Übergang in die Busruhe vollzogen haben und das Universalgateway muss das Senden eigener Botschaften einstellen. Andernfalls wird durch den weiteren Versandt von Botschaften der Bus wieder geweckt und das Fahrzeug kann die Anforderungen an den Ruhestrom nicht einhalten. Wird die CAN-Schnittstelle ausschließlich zum Senden von Werten an ein Messsystem genutzt, ist kein Netzwerkmanagement notwendig. In der derzeitigen Konfiguration des Universalgateways werden lediglich Werte, welche über den WBus von einem Heizgerät abgefragt werden, über den CAN weitergeleitet. Aus diesem Grund ist derzeit kein Netzwerkmanagement implementiert.

Der **CAN-Router** arbeitet ähnlich wie der Router für den WBus. Anhand des Empfangs-Identifier wird ermittelt, auf welchen CAN diese Botschaft weitergeleitet werden soll. Für jeden Ziel-CAN ist ein Bereich im Array (mehrdimensionales Array) realisiert, in welchem die Identifier definiert sind, welche auf diesen Bus geroutet werden sollen. Der erste Eintrag dieses Arrays gibt die Anzahl der enthaltenen Botschafts-Identifier an. Es folgt anschließend je Identifier ein 32-Bit-Wert, damit Standard- und Extendet-Identifier geroutet werden können. Die Konfiguration dieses Arrays erfolgt im Header-File „can_Router_Cfg.h“.
Wird eine Botschaft empfangen, wird der Empfangs-Identifier mit jedem Eintrag in diesem Array verglichen. Ist der Identifier enthalten, dann wird die Botschaft auf dem Ziel-CAN versandt (canh_SendRequest() an den CAN-Handler). Ist die Botschaft nicht im Array aufgeführt, wird diese automatisch über die Funktion cans_HandleRxService() an den Signal-Manager weitergeleitet.

Innerhalb des CAN-Signal-Managers wird der Inhalt der empfangenen Botschaft dekodiert, in Strukturen gepuffert und der Empfang der Botschaft an die RTE gemeldet (rte_CanService_Notification(ID)). Die Strukturen, in welchem die empfangenen Daten abgelegt werden, entsprechen dabei exakt dem Aufbau der CAN-Botschaft. Auf diese Weise ist ein einfacher Zugriff auf die einzelnen Signale möglich (z.B. mMotor1.Motordrehzahl).
Im Regelfall enthält eine CAN-Botschaft deutlich mehr Informationen, als benötigt werden. Ferner werden einige Signale durch Zusatzinformationen plausibilisiert (z.B. durch Gültigkeitsbits oder Botschaftszähler). Die Auswertung und Plausibilisierung der empfangenen Signale erfolgt durch die RTE, indem diese nach der Meldung einer empfangenen Botschaft die benötigten Signale der Botschaft abfragt und als logisches Signal ablegt.

Beispiel für die Plausibilisierung der Motordrehzahl und Überführung in ein logisches Signal:

CAN.Motordrehzahl = CAN-Signal mMotor1.Motordrehzahl,
wenn:

- mMotor1.Motordrehzahl ≠ Fehlerwert und
- Zündung aktiv

sonst:

- CAN.Motordrehzahl = 0

5.8.4 Routen von Daten auf dem CAN

Das Routen von Informationen von einem auf einen anderen CAN ist nicht weiter problematisch. Anders als beim WBus sind die Laufzeiten der Botschaften auf dem CAN sehr kurz. Bei einer Bitrate von 500 kbit/s beträgt die maximale Übertragungsdauer einer Botschaft mit 11 Bit-Identifier 225 µs. Eine Botschaft mit 29 Bit-Identifier benötigt maximal 260µs. [3] Ferner handelt es sich beim CAN um ein Broadcast-System, bei dem auf den Versandt einer Botschaft keine Antwort erwartet wird.
Beim Routing von CAN-Botschaften kann der Inhalt der nicht verändert oder ergänzt werden (OSI-Schicht 3). Ist es erforderlich, dass Daten manipuliert oder ergänzt werden (z.B. wenn ein Signal in eine andere Botschaft umgesetzt werden soll), dann muss das Routing innerhalb der Applikation oder der RTE erfolgen (OSI-Schicht 7).
Werden zwei Busse miteinander über ein Gateway verbunden, ist es üblich, dass nicht alle Botschaften zwischen den Busse geroutet werden. Meist werden die auf dem Ziel-Bus benötigten Signale in eine Botschaft zusammengefasst. Auf diese Weise kann die zur Verfügung stehende Bandbreite des CAN optimal genutzt werden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

5.9 RTE (Laufzeitumgebung)

Die RTE wurde als Verbindung zwischen der Applikation und der Basis-Software realisiert (Laufzeitumgebung). Die Basis-Software wird gegenüber der Applikation abstrahiert. Alle Informationen der Basis-Software, welche die Applikation benötigt, werden in der RTE zwischengespeichert.

Zur Aufnahme der Daten innerhalb der RTE wurden statische Strukturen angelegt. Es wurde je eine Struktur für die lokalen Ein- und Ausgänge des Gateways, die Daten vom WBus und die Daten vom CAN realisiert. Die Basis-Software arbeitet weitgehend eigenständig. Sind für die RTE relevante Daten aufbereitet (z.B. ein Pegelwechsel an einem digitalen Eingang wurde erkannt), dann wird diese Information an die RTE übergeben. Der Zugriff auf die Daten erfolgt dabei sowohl von der Basis-Software als auch von der Applikation über folgende Funktionen:

- rte_ReadIO() / rte_WriteIO()
- rte_ReadWbus() / rte_WriteWbus()
- rte_ReadCan() / rte_WriteCan()

Die Adressierung der Daten erfolgt über symbolische Bezeichner, welche innerhalb des Header-Files der RTE global in Form von Enumerationen definiert wurden. Es wurde je ein Satz absoluter Bezeichner für den Zugriff der Basis-Software und ein Satz relativer Bezeichnern für den Zugriff der Applikation definiert. Die Bezeichner, welche durch die Basis-Software genutzt werden, sind unveränderlich (z.B. DigIn0). Bezeichner, welche zum Zugriff der Applikation dienen, können den Erfordernissen angepasst werden. Wird beispielsweise über den digitalen Ausgang 1 ein Ventil als Aktor angesteuert, dann kann als symbolischer Bezeichner der Name „Ventil“ zum Ansteuern genutzt werden. Auf die Funktion der Basis-Software hat dies keinen Einfluss. Für den Schreibzugriff auf die Daten der RTE gelten dieselben Regeln wie für den Zugriff auf globale Variablen. Die Daten dürfen nur von einer Seite beschrieben werden. Welches Modul der Software die Daten der RTE beschreibt ist als Kommentar zur Definition der Bezeichner im Header-File vermerkt.

Neben dem Puffern von Daten kann die RTE das Senden von Botschaften auf dem CAN oder die Abfrage von Daten über den WBus initiieren. Ferner wird innerhalb der RTE die TimeOut-Überwachung der CAN-Botschaften realisiert. Für diese Aufgabe wurde für die RTE ein eigener Task konfiguriert, welcher zyklisch alle 20 ms vom Betriebssystem aufgerufen wird. In der aktuellen Konfiguration des Universalgateways steuert die RTE die zyklische Abfrage (1 x je Sekunde) der Messdaten von der Standheizung über den WBus und deren Versandt über den CAN.

5.10 Beispielapplikation

Zur Darstellung der Funktionalitäten des Universalgateways wurde eine Beispielapplikation erstellt. Anforderung war es, zyklisch über den WBus, Daten von einem Standheizungs-Steuergerät auszulesen und auf dem Display verteilt auf mehrere Bildschirme anzuzeigen. Ferner sollen die abgefragten Daten auf dem CAN an ein Messsystem weitergeleitet werden. Auf diese Weise soll die Nutzung des Gateways als eigenständiges Diagnoseinstrument dargestellt werden.

Für die Abfrage der Daten von der Standheizung wird der WBus 1 genutzt. Das Universalgateway simuliert dazu den Diagnosetester. Der Akzeptanzfilter des WBus-Treibers (siehe Kapitel 5.8.1) wurde so konfiguriert, dass alle Daten, welche von der Standheizung an den Diagnosetester adressiert werden, durch das Gateway akzeptiert und verarbeitet werden. Für die zyklische Abfrage der Daten wurde innerhalb der RTE eine Ablaufsteuerung realisiert. Einmal je Sekunde wird die Abfrage angestoßen. Für die Abfrage aller benötigten Daten werden insgesamt sechs verschiedene Anfragen an die Standheizung benötigt. Die Ablaufsteuerung wurde innerhalb der Funktion „rte_GetHgInfo()“ durch einen statisch deklarierten Zähler und einer Mehrfachauswahl realisiert. Der Zählerstand verweist dabei auf den aktuell abzufragenden Wert. Wird die Antwort auf eine Datenanfrage von der Standheizung empfangen, wird automatisch der Zähler inkrementiert und die nächste Datenanfrage versandt, solange, bis alle benötigten Daten (z.B. Messwerte, Status der Ausgänge) ermittelt wurden. Nach Ablauf einer Sekunde wird diese Sequenz mit dem Zählerstand Null erneut gestartet. Reagiert die Standheizung nicht, weil diese z.B. nicht angeschlossen ist, dann werden die einzelnen Datenanfragen nacheinander im Sekundentakt versandt, solange, bis eine gültige Antwort empfangen wurde. Für das Zusammenstellen der Frames und die Entschlüsselung der empfangenen Diagnosedaten wurde innerhalb der WBus-Services die Zusammensetzung der entsprechenden Frames gemäß der WBus-Spezifikation realisiert [19]. Neben der Abfrage von Diagnosedaten wurde der Wbus 0 für den Anschluss eines Funkempfängers konfiguriert, um die Standheizung steuern zu können (Konfiguration siehe Abbildung 59). Die Kommunikation zwischen dem Funkempfänger und der Standheizung erfolgt dabei über das Gateway. Für das Routing der entsprechenden Frames wurde die Routing-Tabelle so konfiguriert, dass Frames, welche auf dem WBus 0 an die Standheizung adressiert werden, automatisch auf den WBus 1 und die Antwort wieder zurück auf den WBus 0 gespiegelt werden. In Abbildung 53 bis Abbildung 55 ist das Routing des Befehls zum Abschalten der Standheizung sowie dessen Antwort dargestellt.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

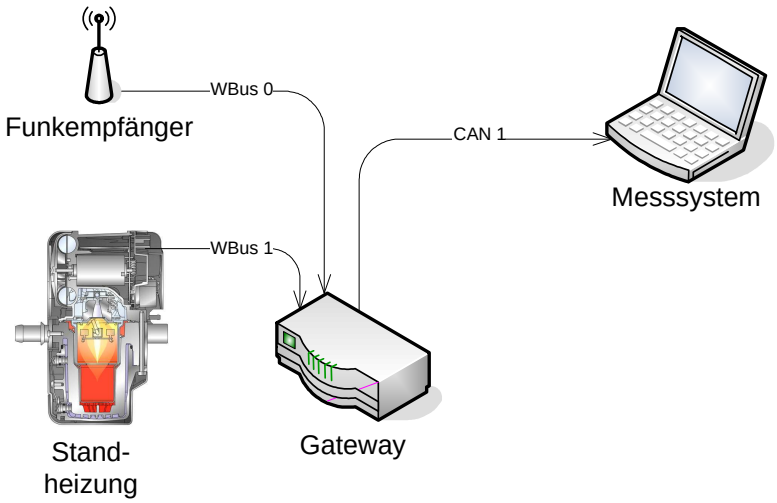


Abbildung 59: Konfiguration der Beispielapplikation

Die von der Standheizung zyklisch abgefragten Daten werden an die RTE übergeben. Diese legt die Daten in einer entsprechend konfigurierten statischen Struktur für den Zugriff der Applikation ab.

Die Anwendungsebene selbst ist in die Bereiche Applikation und LCD-Applikation aufgeteilt. Jede dieser Teilapplikationen besitzt einen eigenen zyklischen Task für die Erfüllung Ihrer Aufgaben.

5.10.1 LCD-Applikation

Die LCD-Applikation übernimmt die komplette grafische Darstellung der Diagnosedaten der Standheizung und ist weitgehend unabhängig von der restlichen Applikation angelegt. Nach dem Abschluss der Initialisierung des LCD wird der Startbildschirm mit dem Firmenlogo angezeigt (siehe Abbildung 60).

Abbildung 60: Startbildschirm

Die Größe der Grafik für den Startbildschirm wurde so knapp wie möglich bemessen und auf einem weißen Bildschirm dargestellt um möglichst wenig Speicherplatz zu benötigen.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Die erforderlichen Daten wurden mit dem Grafik-Konverter (siehe Kapitel 5.7) in ein Daten-Array konvertiert und als Header-File in die LCD-Applikation eingebunden. Die Grafik benötigt bei einer Größe von 170 x 46 Pixeln und einer Auswahl von 256 Farben einen Speicherplatz von 3677 Byte.
Nach Ablauf der 2,5 Sekunden zum Darstellen des Startbildschirms wechselt die Anzeige auf den ersten Bildschirm der Diagnosedaten (siehe Abbildung 61). Die Daten wurden dabei in Anlehnung an die Darstellung im Diagnosetester des PCs dargestellt.

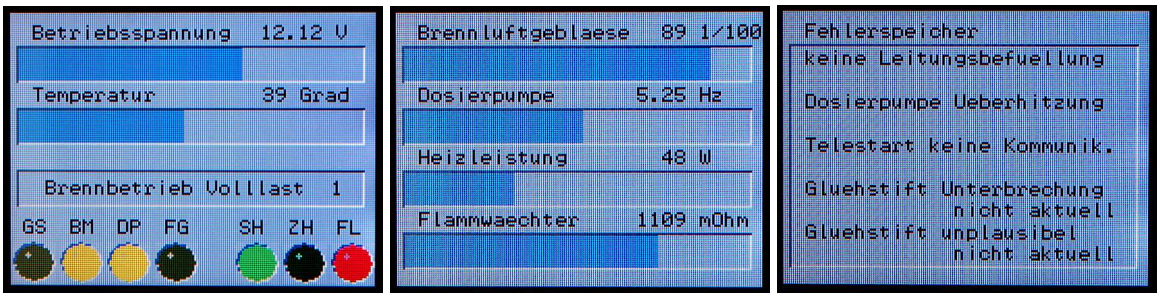


Abbildung 61: Darstellung der Diagnosedaten

Auf dem letzten Bildschirm wurden die Zustände der analogen und digitalen Eingänge des Universalgateways dargestellt (siehe Abbildung 62). Für die digitalen Ein- und Ausgänge wurden die Kontrollleuchten der Graphic-Services verwendet. Für die digitalen Ausgänge kann dabei der vom IO-Manager ermittelte Zustand (Aus = dunkelgrün, Ein = hellgrün, Unterbrechung = hellgelb und Kurzschluss = hellrot) angezeigt werden.

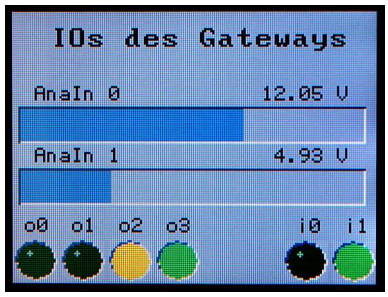


Abbildung 62: Darstellung der Ein- und Ausgänge

Bis auf den Startbildschirm und die Kontrollleuchten werden alle Diagnosedaten über Funktionen der Graphic-Services (Text, Bargraph, Frame) realisiert (siehe Kapitel 5.7). Auf diese Weise wird sehr wenig Speicherplatz für die Darstellung der Diagnosedaten benötigt.

Durch das Betätigen von „Button 0“ kann zwischen den einzelnen Bildschirmen gewechselt werden. Ist der letzte Bildschirm erreicht, dann wird automatisch zum ersten Diagnosebildschirm gewechselt. Zur Steuerung des Bildschirmaufbaus kommt eine einfache Ablaufsteue-

rung zum Einsatz, welche durch die Betätigung des „Button 0“ in den nächsten Zustand wechselt (Auswertung der Flanke 0 -> 1). Bei einem Zustandswechsel werden zuerst alle Elemente dargestellt, welche nicht veränderlich sind (z.B. Überschriften und Rahmen). Die Anzeige der Diagnosedaten erfolgt dann innerhalb eines zyklischen Task (100 ms). Beim Start des Task wird geprüft, welcher Bildschirm derzeit zur Anzeige kommt. Die LCD-Applikation fragt die aktuellen Daten für die Anzeige von der RTE ab und prüft, ob sich diese seit dem letzten Task verändert haben. Wird ein geänderter Wert erkannt, wird die Darstellung dieses Wertes auf dem LCD angepasst. Alle anderen Werte bleiben unverändert. Der Programmablaufplan für die LCD-Darstellung ist in Anhang C dargestellt. Vom LCD können keine Daten abgefragt werden, aus diesem Grund müssen die dargestellten Werte für den Vergleich mit den neuen Werten innerhalb der LCD-Applikation in statischen Strukturen gepuffert werden (je Bildschirm wurde ein Puffer angelegt). Da so nur Daten zum LCD übertragen werden, wenn sich diese ändern und der LCD-Task mit einer geringen Priorität als präemptiv konfiguriert wurde, wird der Programmablauf des restlichen Systems nicht weiter beeinflusst.

5.10.2 Applikation

Die eigentliche Applikation ist sehr einfach gehalten. Es werden lediglich die Zustände der digitalen Eingänge und der Zustand der Button 1 und 2 ausgewertet. Erkennt die Applikation an Buton 1 oder 2 einen Flankenwechsel, wird der Zustand der digitalen Ausgänge 0 und 1 gewechselt. Die digitalen Ausgänge 2 und 3 werden entsprechend dem Pegel an den digitalen Eingängen 0 und 1 geschaltet. Die Zustände der Ein- und Ausgänge können über den Bildschirm 4 auf dem LCD beobachtet werden.

Eine weitere Funktion, welche innerhalb der Applikation realisiert wurde, ist das Routing der zyklisch ermittelten Diagnose-Daten vom WBus auf den CAN. Für jedes WBus-Signal wurde innerhalb der Software CANoe der Firma Vector ein CAN-Signal generiert. Basis für die CAN-Signale bildet die Anordnung und Skalierung der Daten auf dem WBus, um Ungenauigkeiten beim Umrechnen der Formate zu vermeiden. Für die Übertragung der Daten wurde der CAN 1 mit einer Übertragungsgeschwindigkeit von 500 kbit/s konfiguriert.
Immer wenn eine Antwort der Standheizung auf die Anfrage von Diagnose-Daten vom Universal-Gateway empfangen wurde, ruft der WBus-Router die Callback-Funktion der RTE auf, um den Eingang von Daten zu melden. Die RTE informiert ihrerseits die Applikation, welche die Daten vom WBus in eine CAN-Botschaft verpackt und eine Sendeaufforderung auslöst. Auf diese Weise werden alle Diagnosedaten zyklisch an ein Messsystem zur Aufzeichnung

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

übertragen. Für die Darstellung der CAN-Daten (siehe Abbildung 63 und Abbildung 64) kam dabei die Software CANoe der Firma Vector zum Einsatz.

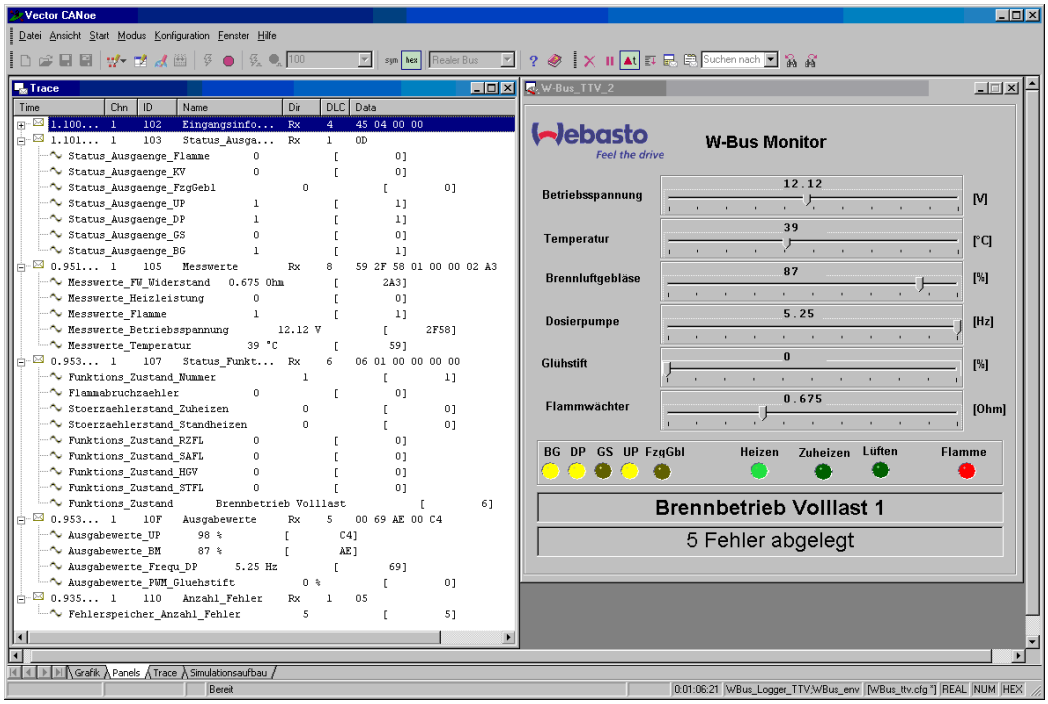


Abbildung 63: Darstellung der WBus-Daten auf dem CAN

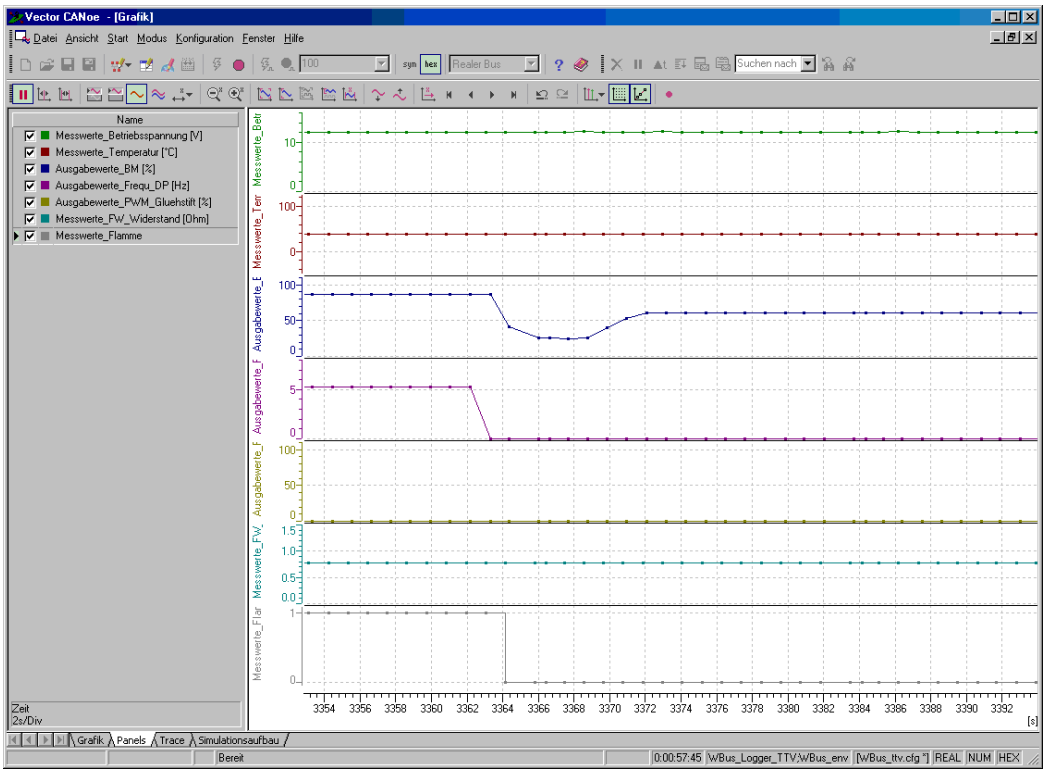


Abbildung 64: Grafische Darstellung der Messdaten auf dem CAN

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

6 Validierung

Innerhalb der Testphase wird nachgewiesen, dass die an das System gestellten Anforderungen erfüllt werden. Dieser Nachweis wird in mehreren Stufen durch eine umfangreiche Validierung auf Modul-, Integrations- und Systemebene erbracht (siehe Kapitel 2.2). Für eine komplette Validierung muss es zu jeder Anforderung der Spezifikation mindestens einen Testfall geben [7].

Auf Grund der engen Zeitschiene beschränkte sich die Validierung des Universalgateways während der Zeit der Diplomarbeit weitestgehend auf den Test der Funktionen und Module.
Eine komplette Validierung der Software des Universalgateways erfolgt, wenn die endgültige Hardwareumgebung zur Verfügung steht und die Module an die entsprechenden Anforderungen angepasst wurden.

6.1 Modultest

Jede Funktion und jedes Modul, welches im Rahmen der Diplomarbeit erstellt wurde, wurde gegen die in Kapitel 4 gestellten Anforderungen geprüft.
Diese Tests orientierten sich dabei an folgenden Fragen:

- Erfüllt das die Funktion alle Anforderungen der Spezifikation?
- Werden alle Fehlermöglichkeiten beim Aufruf einer Funktion abgefangen?
- Läuft die Funktion auch an den Grenzen stabil?

Bei vielen Modulen genügten einfache Tests, um die korrekte Funktion zu überprüfen. Der Test komplexer Zustandsautomaten hingegen erforderte einen deutlich höheren Testaufwand. Funktionen, welche die Basis für andere Funktionen bilden (z.B. Treiber) wurden dabei intensiver getestet, um im späteren Verlauf der Entwicklung verdeckte Fehler zu vermeiden. Die in den Modultest des Universalgateways erkannten Fehlfunktionen wurden parallel zum Test abgestellt.

6.2 Integrationstest

Wurde ein Modul, welches aus mehreren Einzelfunktionen besteht, fertiggestellt (z.B. die WBus-Kommunikation), dann wurde das gesamte Modul einem Integrationstest unterzogen. Die durchgeführten Tests orientierten sich dabei an denselben Kriterien, wie die Modultests.
Durch die durchgeführten Integrationstest wurden Schwachstellen beim Routen von langen WBus-Frames zwischen den Bussen festgestellt (siehe Kapitel 5.8.2). Innerhalb der Weiterentwicklung des Projektes bilden diese Erkenntnisse die Basis für das künftige Vorgehen.

7 Zusammenfassung und Bewertung des Entwicklungsziels

Ziel dieser Diplomarbeit war es, ein Universalgateway als Rapid-Prototyping-System und Diagnoseinstrument zu spezifizieren und ein Funktionsmuster zu erstellen. Diese Diplomarbeit bildet dabei Basis für eine weitergehende Entwicklung des Systems.

Im theoretischen Teil der Diplomarbeit wurden in Zusammenarbeit mit den Entwicklungsteams und den Automobilherstellern die Anforderungen an das zu entwickelnde Universalgateway ermittelt (siehe Kapitel 3). Aus diesen Anforderungen erfolgte in Kapitel 4 der Systementwurf und die Definition der Softwarearchitektur. Ferner wurden die Anforderungen an die Hardware beschrieben. Die Umsetzung der spezifizierten Anforderungen und deren Besonderheiten wurden in Kapitel 5 beschrieben. Für die Umsetzung kam ein Evaluation-Board mit V850 Fx3-Prozessor der Firma NEC zum Einsatz, welches entsprechend der ermittelten Hardwareanforderungen, um die für das Projekt notwendigen Komponenten wie beispielsweise digitale und analoge Eingänge sowie einem LCD erweitert wurde.
Nach der Konfiguration des OSEK-Betriebssystems und der vorhandenen AUTOSAR-Treiber wurden diese integriert und nacheinander die einzelnen Module der Software erstellt. Als Beispielapplikation für das Universalgateway wurde die Abfrage und Anzeige von Diagnose-Daten (z.B. Temperaturen, Drehzahl, Fehlerspeicher) einer Standheizung sowie das Routing dieser Daten auf den CAN realisiert. Ferner wurden auf einem separaten Bildschirm die Zustände der digitalen und analogen Ein- und Ausgänge zur Anzeige gebracht.

Während der Entwicklung der Module zur Kommunikation über den Webasto-eigenen Bus wurden beim Test der Systemgrenzen Einschränkungen festgestellt. Das Routing ganzer Datenrahmen auf dem Weasto-eigenen Bus ist nur bis zu einer Länge von 30 Byte sinnvoll möglich ist. Ursache für diese Einschränkung ist die Laufzeit der Signale auf den Bussen. Ab einer Länge der Datenrahmen von ca. 30 Byte wird die maximale Antwortzeit vom Versenden der Anfrage durch den bis zum Empfang der Antwort überschritten.

Mit dem aktuellen Entwicklungsstand steht ein Universalgateway als Funktionsmuster zur Verfügung, welches dieselbe Hardwareplattform wie die aktuellen Standheizungsprojekte besitzt und eine an AUTOSAR angelehnte Softwarearchitektur besitzt. Durch die Abstraktion über die Laufzeitumgebung (RTE) ist der Austausch von Modulen zu anderen Projekten möglich.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

8 Ausblick

Mit dem zur Verfügung stehenden Funktionsmuster des Universalgateways besteht die Grundlage für eine Weiterentwicklung bis zum Einsatz als Rapid-Prototyping-System im Fahrzeug. Im nächsten Schritt muss dazu eine seriennahe Hardware entwickelt und erstellt werden, auf der die noch fehlenden Softwaremodule wie z.B. der NV-Memory-Manager realisiert werden können.

Die bisher durchgeführten Integrationstests haben im Grenzbereich Schwächen im WBus-Handler aufgezeigt. Die Zeit bis zur Fertigstellung der endgültigen Hardware kann daher genutzt werden, um den aktuellen Stand der Software noch intensiver zu testen und bestehende Module zu verbessern. Ferner besteht das Problem, dass auf dem Webasto-eigenen Bus Datenrahmen mit mehr als 30 Byte nicht sicher geroutet werden können.
Eine mögliche Lösung besteht darin, in der Spezifikation des Webasto-Busses eine Beschränkung auf 30 Byte einzufügen. Ist dieses nicht möglich, kann das Routing längerer Datenrahmen nicht mehr auf OSI-Schicht 3 erfolgen. In diesem Fall müssen diese aktiv durch die Laufzeitumgebung oder die Applikation bearbeitet werden (oberhalb von OSI-Schicht 7).

Eine weitere Verbesserung wäre bei Aufbereitung der CAN-Daten für die Integration möglich. Die manuelle Erstellung der Signalfestlegungen für den CAN ist sehr aufwendig und fehlerträchtig. Für jede CAN-Botschaft, welche direkt durch das Universalgateway ausgewertet oder versandt werden soll, muss eine entsprechende Struktur für die Aufnahme der Daten erstellt werden. Weiterhin muss jeweils der CAN-Signalmanager so angepasst werden, dass die Strukturen korrekt mit Informationen gefüllt werden. Für eine komfortable Konfiguration CAN-Signale wäre es daher wünschenswert, ein Tool für den PC zu erstellen, welches die Signale einer Datenbasis für den CAN so aufbereitet, dass diese auf einfache Weise eingebunden werden können.

Neben der Nutzung des Universalgateways als Rapid-Prototyping-System wäre es bei entsprechender Größe der Zielhardware möglich, dieses System auch als handliches rechnerunabhängiges Diagnosesystem einzusetzen. Auf diese Weise könnten in Erprobungsfahrzeugen permanent die wichtigsten Daten der Standheizung auf einem Display dargestellt werden.

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Literaturverzeichnis

[1] Mutz, Martin: Eine durchgängige modellbasierte Entwurfsmethodik für eingebettete Systeme im Automobilbereich
Dissertation – Poing: Cuvillier Verlag, 2005

[2] Grzemba, Andreas; von der Wense, Hans-Christian: LIN-Bus
1. Auflage – Poing: Franzis-Verlag, 2005

[3] Zimmermann, Werner; Schmidgall, Ralf: Bussysteme in der Fahrzeugtechnik
3. Auflage – Wiesbaden: Vieweg+Teubner, 2008

[4] Elektronik Automotive; LIN Special 2004
Poing: WEKA Fachzeitschriften–Verlag 2004

[5] Müller, Marcus: Methode zur automatischen Prüfung von Netzwerken in Fahrzeugen
Dissertation – Technische Universität Braunschweig, 2002

[6] Borgeest, Kai: Elektronik in der Fahrzeugtechnik
1. Auflage – Wiesbaden: Vieweg–Verlag, 2008

[7] Schäuffele, Jörg; Zurawka, Thomas: Automotive Software Engineering
3. Auflage – Wiesbaden: Vieweg–Verlag, 2006

[8] Reif, Konrad; Automobilelektronik
- 2. Auflage – Wiesbaden: Vieweg–Verlag, 2007

[9] Wietzke, Joachim; Tien Tran, Manh: Automotive Embedded Systeme
1. Auflage – Berlin: Springer-Verlag, 2005

[10] DaimlerChrysler AG: Ausführungsvorschrift A 204 000 77 99 OSEK Design Guide
Version 12 – Stuttgart: 2003

[11] 3SOFT GmbH: ProOSEK UserGuide;
Build–Version 20070717, Revision 1.35 – Erlangen: 2007

[12] 3SOFT GmbH: ProOSEK Architecture Notes for the NEC V850 Achitecture;
Build–Version 20070717, Revision 1.17 – Erlangen:, 2007

[13] AUTOSAR GbR: Layered Software Architecture
Document Version 2.1.0, Release 2.1, Revision0014; 2007

[14] AUTOSAR GbR: Specification of Operating System
Document Version 2.1.0, Release 2.1, Revision0014; 2007

[15] AUTOSAR GbR: Specification of I/O Hardware Abstraction
Document Version 1.1.1, Release 2.1, Revision0015; 2007

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

[16] AUTOSAR GbR: Specification of Platform Types
Document Version 2.1.0, Release 2.1, Revision0014; 2007

[17] AUTOSAR GbR:; Specification of Standard Types
Document Version 1.1.1, Release 2.1, Revision0014; 2007

[18] AUTOSAR GbR: Specification of RTE
Document Version 1.1.1, Release 2.1, Revision0014; 2007

[19] Webasto AG: Lastenheft Bidirektionale serielle Schnittstelle W-Bus
Version 4.03.6 – Stockdorf: Webasto AG, 2008

[20] Webasto AG: Programmierrichtlinien
Version: 4.13 – Stockdorf: Webasto AG, 2006

[21] Schober, Stefan; WEC² Software Architecture
Stockdorf: Webasto AG, 08.08.2007

[22] Wabsto AG: Firmware-Pflichtenheft SCS Subsystem VW-Cabrio
Version: 3.2 – Stockdorf:Webasto AG, 2008

[23] NXP: TJA1041 Product data sheet
Rev. 06 – 5. Dezember 2007

[24] Philips: TJA1020 Product specification
Rev. 5 – 13. Januar 2004

[25] Küsters, Peters: Programmierhandbuch für 2,1” Display-Module mit 65536 Farben
Version 1.50 – Willich: WWW.Display3000.com, 23. Januar 2009

[26] NEC: V850ES/Fx3 32-bit Single-Chip Microcontroller User’s Manual
Version 2.2 – Dezember 2007

[27] NEC: AB-050-FX3-P V850ES/Fx3 Starter Board
Version 2.0 – November 2006

[28] KPIT Cummins Infosystems Lt.: Getting Started Document for MCAL Drivers for NEC
Version 2.0.1 – 2007

[29] MISRA-C 2004: Guidelines for the use of the C language in critical systems
Motor Industry Research Association

Chris Kluge
KT07w1
Matrikel-Nr.: 20311

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Anhang A: Matrix der ermittelten Kundenanforderungen

	Kommunikations-Schnittstellen								Diskrete IO										Weckbarkeit		
	CAN			LIN					Digital In		Digital out			Analog In		Analog Out					
	Anzahl High-Speed	Anzahl Low-Speed	Geschwindigkeiten	Anzahl / Version	Geschwindigkeit	Anzahl Most-Schnittstellen	Anzahl Flexray-Schnittstellen	Anzahl Webasto-Buss-Schnittstellen	Anzahl	Schaltschwelle 0/1 bzw. High-active/low/active	Anzahl	Laststrom	High-Side / Low-Side / Relais	Anzahl	Spannungsbereich	Anzahl	Spannungsbereich	Laststrom	Wecken über Datenbus	Wecken über alle Digital In (Flankenwechsel)	Wecken über Wakeup-Leitung
Webasto	0	0	-	0	-	0	0	2	2	keine Info	2	2A	1xhigh, 1x low	2	0-18,5 / 0-32V	0	-	-	ja	ja	nein
Volkswagen	2	Option	100 / 500	2 / V2.1	9,6K / 19,2K	0	0	2	2	L < 2V H ≥ 7V	4	2,5A	2x Brücke 2x high	2	0-17V	0	-	-	ja	ja	nein
Audi	2	Option	100 / 500	2 / V2.1	9,6K / 19,2K	0	0	2	1	L < 2V H ≥ 7V	2	2,5A	high	1	0-17V	0	-	-	ja	ja	nein
BMW	0	0	-	2 / V2.x	9,6K / 19,2K	0	0	1	1	keine Info	1	keine Info	keine Info	1	5-17V	0	-	-	ja	nein	ja
Daimler	1	0	125	0	-	0	0	0	1	keine Info	2	keine Info	high	0	-	0	-	-	ja	nein	ja
Ford	1	0	125 / 500	1 / V2.x	keine Info	0	0	1	1	keine Info	1	1A	keine Info	0	-	0	-	-	ja	nein	nein
Volvo	1	0	125 / 500	1 / V2.x	keine Info	0	0	1	1	keine Info	1	1A	keine Info	0	-	0	-	-	ja	nein	nein
Opel	0	0	-	0	-	0	0	2	2	keine Info	2	2A	high + low	2	0-18,5 / 0-32V	0	-	-	ja	ja	nein
PSA	0	1	0	2 / V2.x	keine Info	0	0	0	2	1xhigh, 1x low	2	keine Info	1xhigh, 1x low	0	0	0	0	0	ja	ja	nein

Chris Kluge
KT07w1
Matrikel-Nr.: 20311

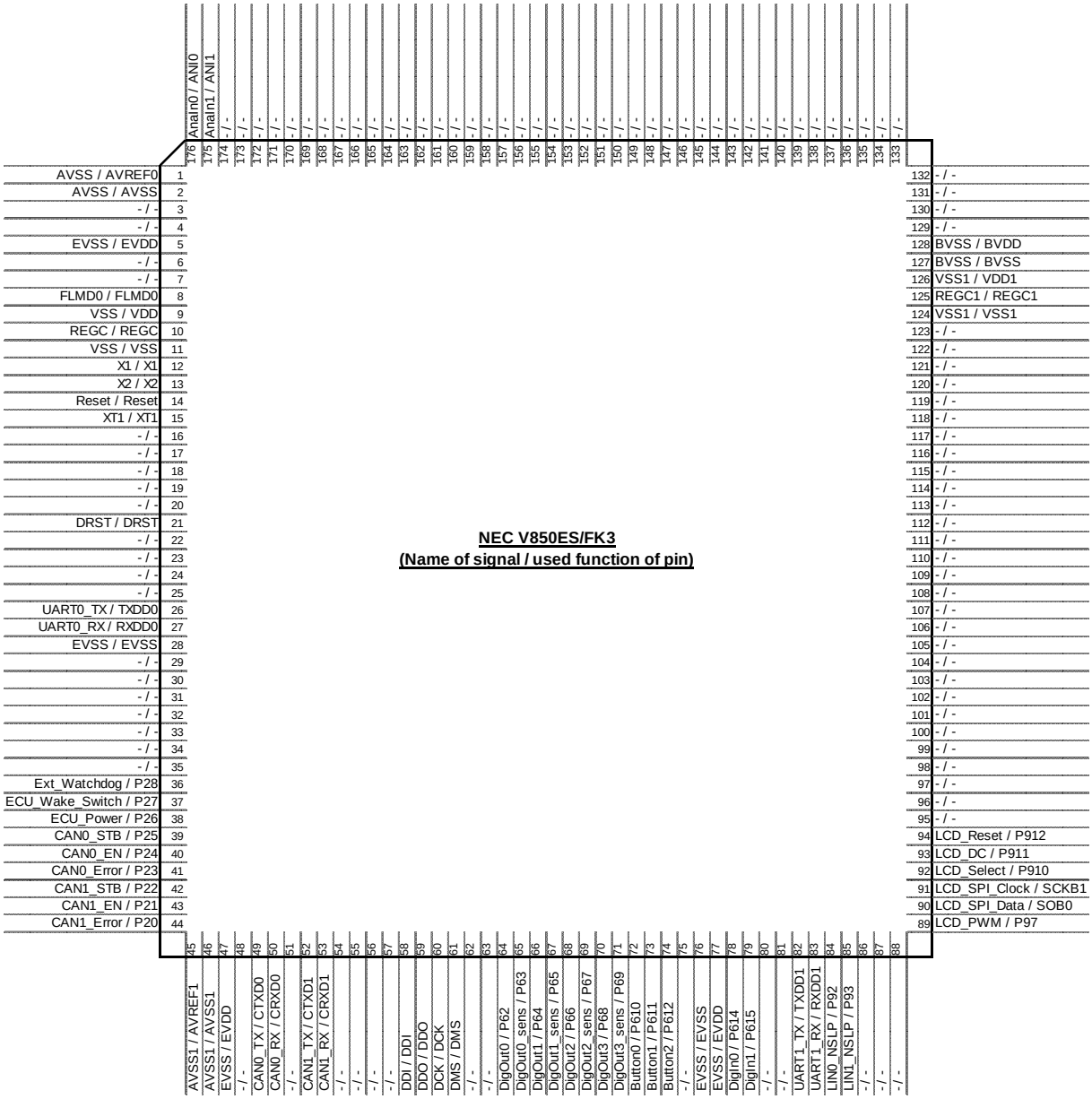
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

	Ruhestrom		Datenlogging			Anzeige / Display				Dignose-Funktionen				Gateway-Funktionen			Programmierbarkeit			
	Ruhestromfähigkeit (dauerhaft an Fzg-Batterie)	Ruhestromaufnahme max.	Datenbusse	Diskrete IO	Interne Werte	Anzeige von einfachen Statusinfos	Anzeige von Bus-Informationen	Grafische Darstellung von Werten	Anzeige von internen Infos	KWP2000	UDS	XCP	Webasto-Bus	Repeater zwischen Bussen	Manipulation von Daten	Senden eigener Botschaften	Frei programmierbar in C	Nutzung von Routinen / Funktionen aus anderen Entwicklungsprojekten	Eingeschränkt parametrierbar (nutzung vorgefertigter Routinen)	Fest konfiguriert für einen Einsatzfall (z.B. als Datenmonitor)
Webasto	ja	100µA	ja	ja	ja	ja	ja	ja	ja	ja	nein	nein	ja	ja	ja	ja	ja	ja	ja	ja
Volkswagen	ja	60µA	ja	ja	ja	ja	ja	ja	ja	ja	ja	nein	ja	ja	ja	ja	ja	ja	nein	ja
Audi	ja	60µA	ja	ja	ja	ja	ja	ja	ja	ja	ja	nein	ja	ja	ja	ja	ja	ja	nein	ja
BMW	ja	100µA	ja	ja	nein	ja	nein	nein	nein	nein	nein	nein	ja	ja	nein	ja	ja	ja	nein	nein
Daimler	ja	60µA	ja	ja	nein	ja	ja	nein	nein	ja	ja	nein	ja	nein	nein	nein	ja	ja	ja	ja
Ford	ja	100uA	ja	nein	ja	ja	ja	Option	ja	ja	Option	ja	ja	nein	nein	ja	Option	ja	ja	ja
Volvo	ja	100uA	ja	nein	ja	ja	ja	Option	ja	ja	Option	ja	ja	nein	nein	ja	Option	ja	ja	ja
Opel	ja	100µA	ja	ja	ja	ja	ja	ja	ja	ja	nein	nein	ja	ja	ja	ja	ja	ja	ja	ja
PSA	ja	keine Info	ja	ja	ja	ja	ja	ja	Option	ja	nein	nein	ja	ja	ja	ja	ja	ja	nein	nein

Chris Kluge
KT07w1
Matrikel-Nr.: 20311

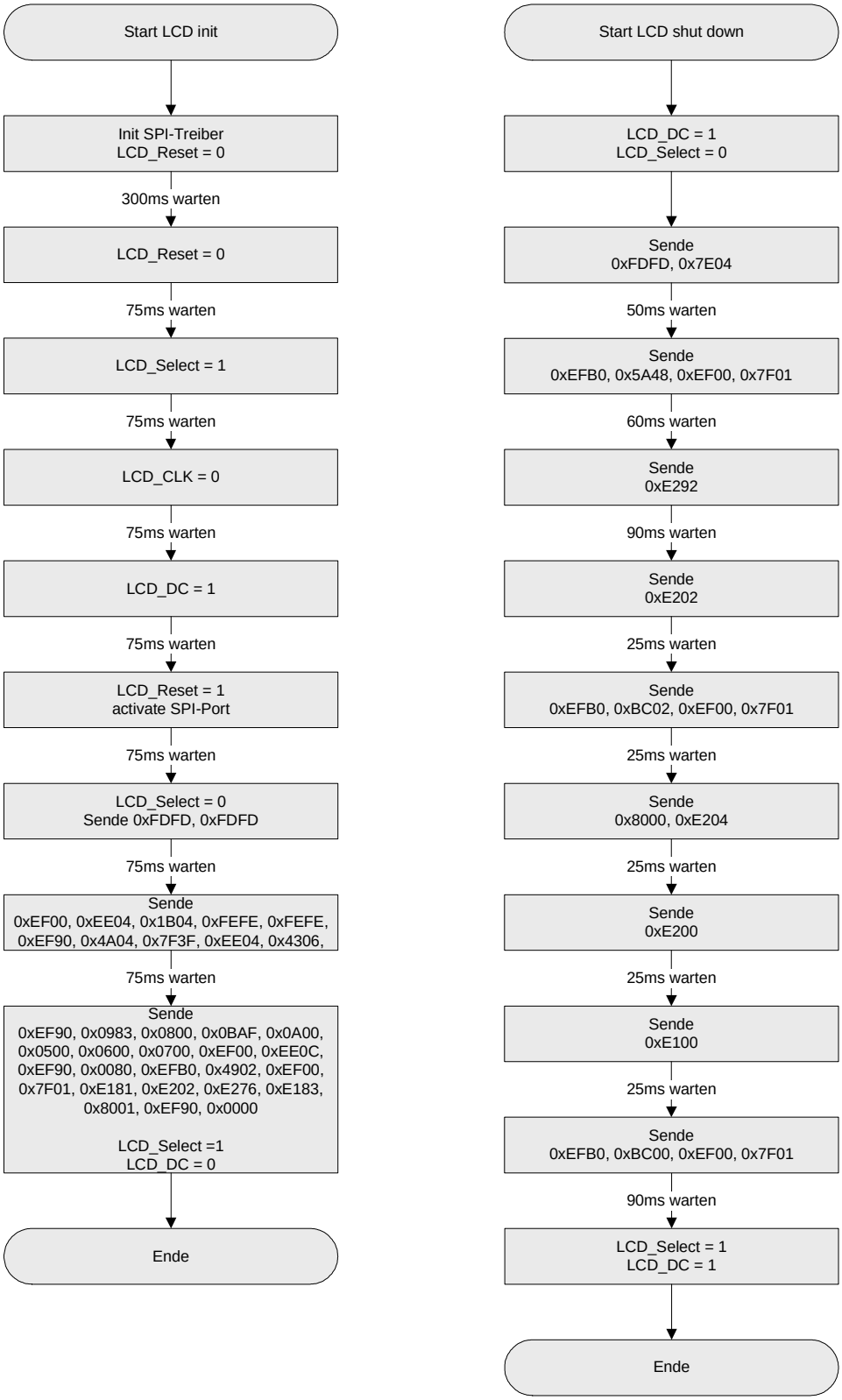
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Anhang B: Pin-Belegung Prozessor



Chris Kluge
KT07w1
Matrikel-Nr.: 20311
Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

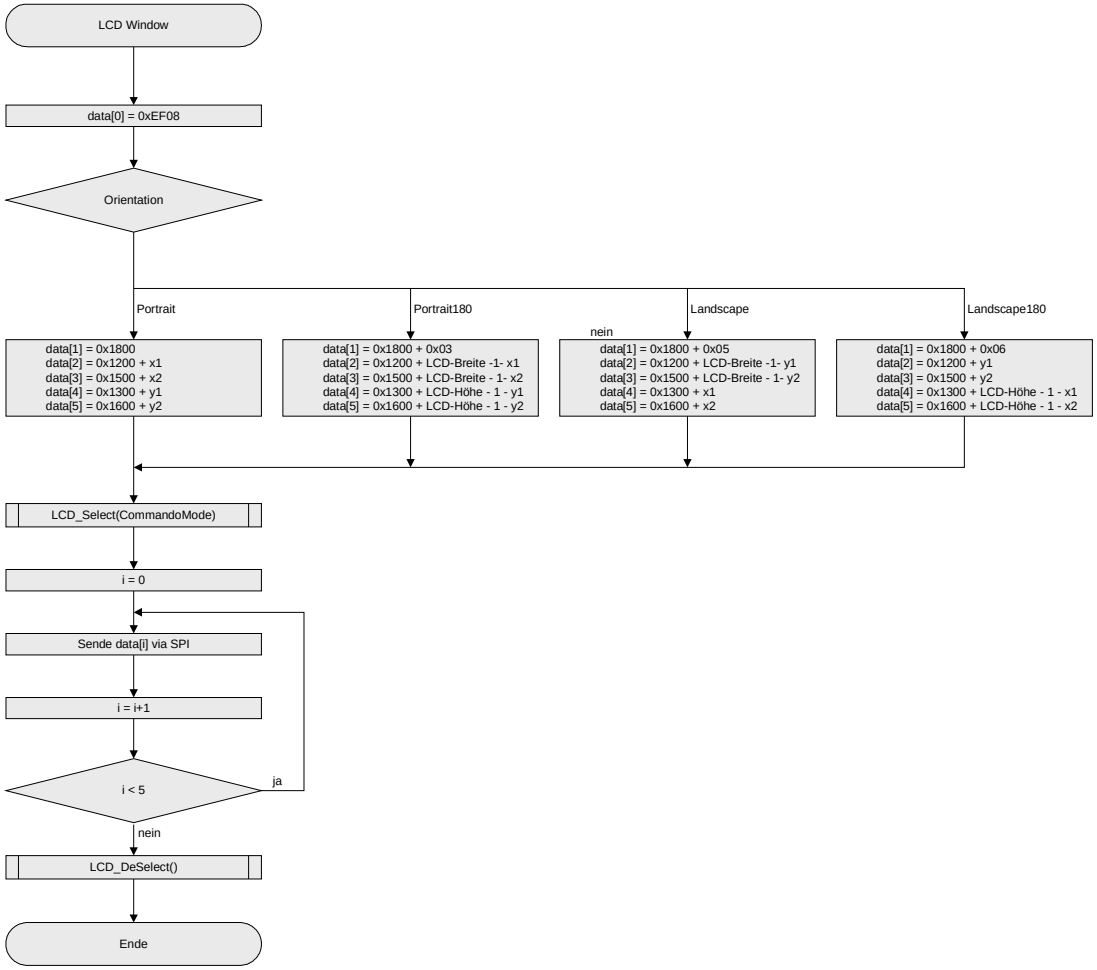
Anhang C: Grundsequenzen für die Kommunikation mit dem LCD



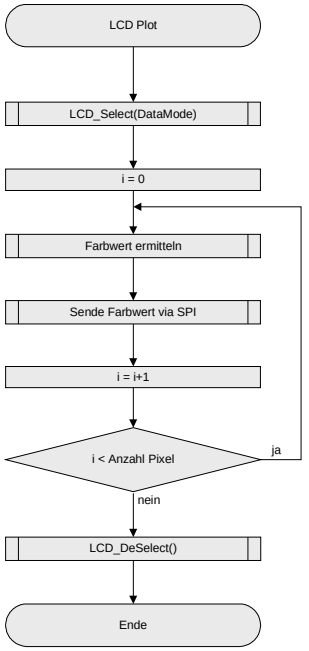
Chris Kluge
KT07w1
Matrikel-Nr.: 20311

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Fenster auf LCD öffnen:



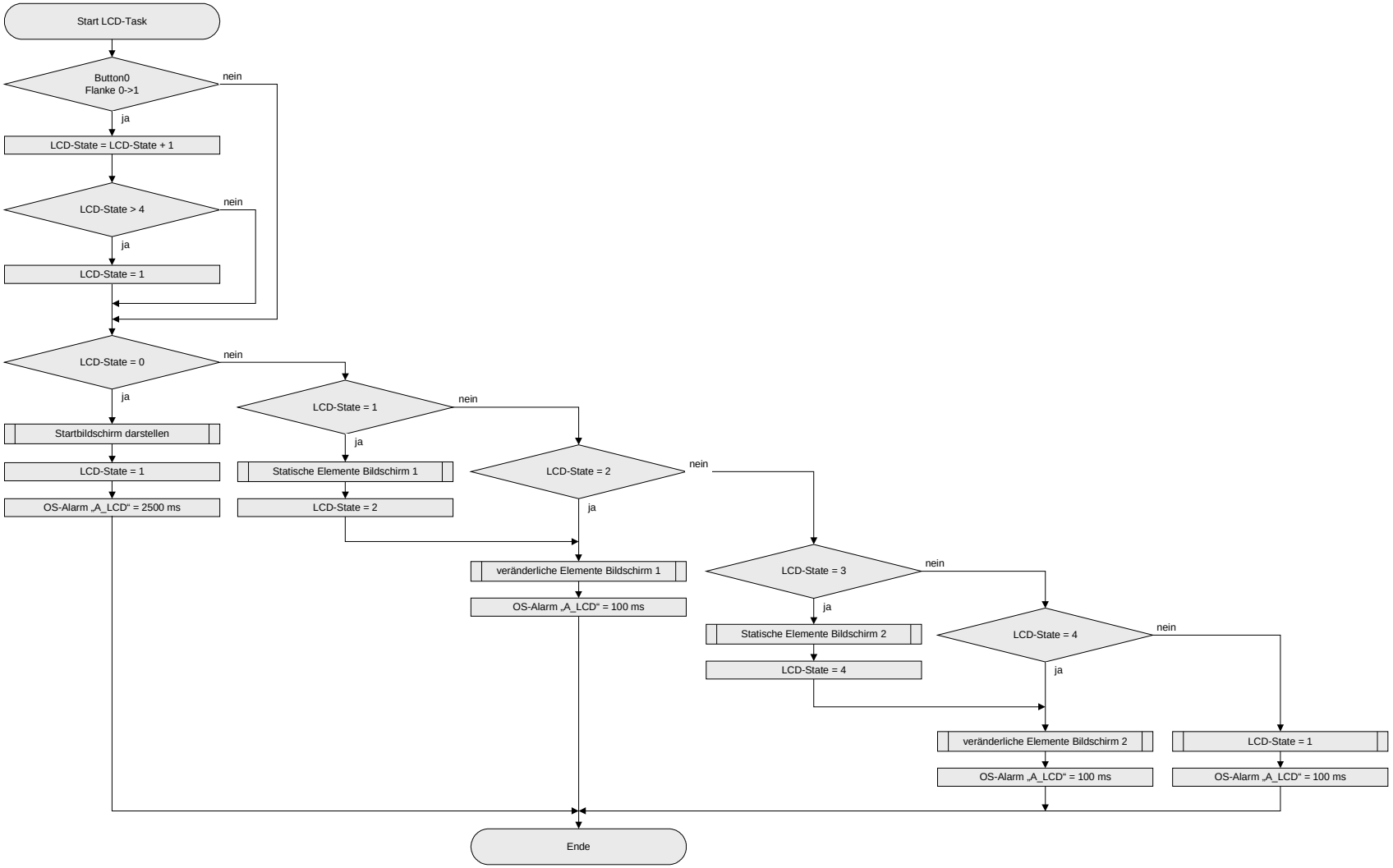
Zeichnen eines Pixels:



Chris Kluge
KT07w1
Matrikel-Nr.: 20311

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Programmablaufplan des LCD-Task (beispielhaft für zwei Bildschirme)



Chris Kluge
KT07w1
Matrikel-Nr.: 20311

Tel.: 0 53 07 / 20 13 55
01 72 / 9 53 64 81
E-Mail: ckluge@htwm.de

Anhang D: Zustandsautomat WBus-Handler